DATABOROUGH

X-ANALYSIS 8

DESIGN RECOVERY CONCEPTS GUIDE

Version 8 Release 7

Beacon House,
South Road, Weybridge,
Surrey, U.K. KT13 9DZ
Telephone:Weybridge (01932) 848564
Facsimile: Weybridge (01932) 859211

Information in this manual is subject to change without notice and does not represent a commitment on the part of Databorough Limited. The software described in this manual is furnished under a license agreement and may be used or copied only in accordance with the terms of the agreement.

Table of Contents

Preface

Developing tools and services for analyzing and reengineering applications for more than twenty years, has given Databorough a unique perspective on the large and complex world of legacy applications running on System i, iSeries and AS/400.

In 2005, IBM and Databorough published an IBM Redbook "Modernizing and Improving the Maintainability of RPG Applications Using X-Analysis Version 5.6". This concepts guide you are reading now expands on the white paper 'Modernizing RPG/COBOL/2E System i applications using X-Analysis 8', incorporating new concepts and methods for design recovery and rebuilding of monolithic RPG/COBOL/2E applications into modern application architectures. Contact info@databorough.com for a copy of the white paper, the Redbook and trial software.

> We will show in this paper how automated component generation from recovered designs can dramatically reduce the costs and risks of an application rewrite and without inheriting the legacy code's redundancy and complexity.

Executive Summary

The knowledge and information contained in an organization's business software is vitally important and extremely valuable but often this information covering the operation, metrics, and design of the software is tantalizingly out of reach. Without this knowledge, maintenance and changes to the system are not as efficient or effective as they could be, and the risk of failure or problems increases exponentially the larger the enhancement required. This could lead to a paralysis where changes can't be made due to a lack of confidence in the outcome.

As many of the systems and software we are discussing here have had a long life and been marketed under various names it is worth making two orientation points:

For consistency throughout this document we will refer to System i as meaning the family of computers that grew out of IBM's System/38 over the last twenty one years namely the AS/400 , iSeries and latterly the System i and IBM i on Power.

Similarly when we refer to the RPG language we will generally mean COBOL, RPG and 2E (When we refer to 2E we mean the CA product and the various incarnations of the Synon software that preceded it).

Accurate and current information about an entire system can greatly improve the productivity of your IT staff, and reduce maintenance costs by eliminating the need to research, catalog and assemble the information manually for each service request, or modernization project.

Existing System i applications whether they are COBOL, RPG or 2E have some fairly consistent and distinct characteristics that mark them out as costly and potentially high-risk:

1. Applications tend to be large and complex
2. Little or no documentation
3. Original Designers and Developers are no longer available
4. They have been developed over many years
5. Monolithic Programming Model
6. Written in obsolete languages

Points 1 through 3 can largely be managed more effectively by investing in a product like X-Analysis to both recover the design of the application, and provide highly productive analysis tooling to compensate for the complexity of the application and the absence of the original development team.

Inconsistent programming standards and designs, significant amounts of redundant and duplicated code, and an increasingly costly demand for globally diminishing legacy development skills, are the results of points 4, 5 and 6.

This concepts guide will illustrate how X-Analysis carries out the Design Recovery process and how it can be used to build re-engineered applications from that Recovered Design.

To fully understand and appreciate the problem domains just think for a minute of two approaches to the above problems namely screen-scraping and code conversion.

Simply screen scraping the user interface with a GUI or web emulation product does not improve the situation, the application may appear slightly more ‘modern’ but the cosmetic changes still leave it with all the same maintenance and enhancement issues and it may be not much easier to use for new users.

The other common approach is code conversion i.e. line by line, syntax conversion of a legacy application, this will typically just transfer the same problems from one environment/language to another. Indeed, it will often produce source code that is less maintainable, effectively canceling out the benefit of using modern technologies and architectures in the first place. Syntax conversions are still being done by some companies and are often promoted by vendors of proprietary development tools for obvious reasons. This approach has never to our knowledge produced an optimum long-term result, despite many attempts over the last two decades.

Removing problems 4 through 6 and thus achieving sustainable and effective application modernization can only really be achieved with an application rewrite or rebuild – which is well recognized of course but usually rejected as not feasible on cost and risk grounds.

We will show in this paper how Design Recovery and automated component generation can dramatically reduce the costs and risks of an application rewrite and without inheriting the legacy code’s redundancy and complexity.

Whatever the approach to modernization, design recovery is the first step. With this understanding, developers can quickly identify the business rules and reusable designs, embedded in core business processes and restructure code, remove dead code, and create reusable components that can be enabled as services within a service-oriented architecture (SOA), or any modern application architecture. This is true, even for companies adopting code generators technologies, as their development environment.

The objective, therefore is a true modernization exercise to extract the essence or design of the legacy application and reuse these designs as appropriate in rebuilding the application, using modern languages, development tools, and techniques, tapping into more widely available skills and resources.

X-Analysis provides analysts, developers, architects and operations teams with detailed analysis and interactive diagrammatic constructs that enable rich understanding of existing applications, whether they were developed yesterday, or 30 years ago. Some companies may have a desire to keep a significant amount of design logic from the existing application design and move that to the modernized version of the application. For those situations, X-Analysis provides design extraction functionality, for automatically creating JEE* & RPG industry standard modern components and constructs as exports from the recovered designs themselves. A legacy application component can be rewritten using a combination of the JSF, EGL, Facelets, and persistence frameworks such as Hibernate, all generated by top-down automation from the recovered X-Analysis model. Because each customer situation is potentially different, the X-Analysis suite is available in different editions that suit the appropriate development stage or budget constraints of each company.

* Jargon Explained:

JEE – Java Extended Edition previously known as J2EE

JSF- JavaServer Faces a web application framework which uses a component-based approach to simplify development of user interfaces for JEE applications.

EGL – Enterprise Generation Language a new high level platform independent language from IBM which produces code which can be compiled into Java or COBOL.

From even the most poorly structured application, the X-Analysis can recover the design logic. Whereas, for more structured applications (e.g. Synon generated applications), X-Analysis can directly extract the details of the existing model, providing an excellent base for efficient and effective design recovery and reengineering to JEE.

Introduction

As we have seen gathering knowledge about System i applications is not a straightforward task for today's generation of business analysts and developers. To illustrate that point and to fully understand the problem domain we will look at Why Design Recovery is difficult by working through the problems that X-Analysis solves in building its repository of design recovery information.

In situations where developers are not familiar with a system or its documentation is inadequate, the system's source code becomes the only reliable source of information. Unfortunately, source code has much more detail than is needed just to understand the system, also it disperses or obscures high-level constructs that would ease the system's understanding. X-Analysis aids system understanding by identifying recurring program features, classifying the system modules based on their purpose and usage patterns, and analyzing dependencies across the modules. This analysis provides detailed design information about the entire system, accessible to non RPG/COBOL/2E experts, and be easily updated to incorporate ongoing changes in the base system.

Whatever the business needs driving companies to modernize their applications, most want to ensure that the business logic and functional design which are core assets to the company, are preserved to varying degrees.

Design Recovery of an application can be broken down into a few logical steps or stages that represent a generic adaptable approach to any application modernization project:

Analysis, Documentation, Application Subdivision – This type of analysis represents the most common use of the X-Analysis tool across the world. On top of very powerful cross-referencing functionality, graphical, narratives or a combination of both, are used to abstract and describe the system in a simple and intuitive way, even for non-RPG/COBOL/2E experts. The legacy application can be completely documented using modern diagramming standards such as UML, Entity Relationship Diagrams, System Flow Diagrams, and Structure Charts etc. Furthermore, the legacy system can be automatically subdivided into application areas so that effective system overview & interface diagrams can be generated. The complete application documentation can then be output to a variety of third party design tools such as Rational, MS Visio, MS Word, etc. – indeed any tool capable of importing XML or DDL is supported.

Recovering an Application design – This advanced level of analysis extracts model information from the existing application. X-Analysis uses its own analysis repository, plus pattern searching algorithms, to derive relational data models, extract business rules, build UML Activity/Use/Case Diagrams, and logical screen flows. Only relevant designs need be used as a base specification for new developers to rewrite the application. The structured, repository-based format of these extracted designs, make it possible, to programmatically reuse them for rebuilding the core of a new application. This can be done with purpose-built tools, with X-Modernize or a combination of both.

Redeveloping Using a Recovered Application Design – This starts with database modernization using the recovered data model. The designs for the view, controller, and business rule logic are also extracted and reused in modern frameworks such as Hibernate, and with new JSF/Facelets and Java bean components. This option makes it possible to programmatically re-factor the existing application into modern, consumable assets and artifacts for developers to use for a system rebuild. The objective is to produce clean, well structured, industry standard code rather than messy syntax conversions with unmaintainable code.

Why Design Recovery is difficult

From the point of view of the user of X-Analysis this process of building the cross-reference repository and deriving the models happens automagically! i.e. Its just there and happens typically as part of the installation process - though it can be triggered again later on if required. However it is worth taking some time to understand this process and to see what happens, how the model is constructed and the relationships inferred.

If you think of a typical System i application it is likely to consist of a mix of RPG programs, DDS files and members for display files, database files and logical views, newer systems may have these interspersed with SQL scripts but the sum of knowledge in that system, how it works and interacts amongst its various elements is contained within those source files and compiled objects - the issue is retrieving that knowledge efficiently.

To understand and fully appreciate the problems X-Analysis solves just consider the process you would have to undertake yourself if you wanted to discover how a system operates or make changes to it. As a simple example for part of your application you have a customer details screen with no dedicated place for an email address and mobile phone numbers, the system has adapted itself to the internet age as many System i apps have done by making use of ‘extra’ and ‘notes’ ad-hoc fields. The system has coped but it has been time consuming to retrieve these details when required for marketing purposes. But there is now a budget to correct this and start to look at modernising the application and making the functionality available to more areas of the business.

You would probably first start by looking at the program and display files that handle the display and maintenance of the customer information, from that you would discover the database tables/files involved.

At this point from a simplistic point of view you have the necessary information to make the changes and they are probably not that difficult - add new fields for email and mobile phone to the database tables or rename the existing ones then modify the program and display files accordingly… but you’re probably thinking what about the rest of the system? What else uses that table? Is the display file used anywhere else? So the change has more aspects than would first appear these are just a few of the questions we have to answer:

- Scope and impact of the change - how many programs and tables are effected?

- Database changes - do we add new fields or just rename the fields and preserve the status quo? Do we know those fields were only used for email and mobile phone data?
- Database integrity - Fields destined for ad-hoc data like 'extra information' and 'notes' are unlikely to have any validation or to be even required so if migrating the existing values to new fields we can't simply copy it over some cleansing will be required.

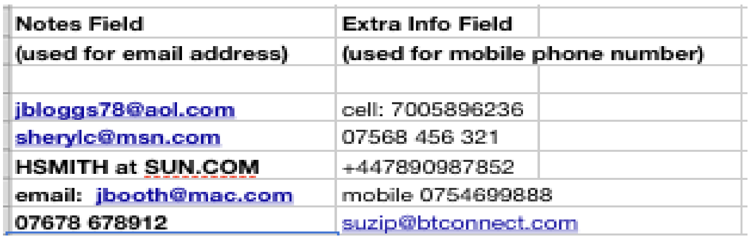
The process of gaining the knowledge to answer these questions may not be all that straightforward, particularly if the systems are complex or the people trying to answer them are new to the application, system or platform.

To assess the scope and impact of the change you need to find out which programs use the files/tables affected , this can be very laborious :

- ✓ Go through all source files in PDM,
- ✓ option 25 to search
- ✓ then F13 to repeat
- ✓ press enter
- ✓ type in your search term
- ✓ review results …

… and that's just the first enquiry! Depending on the complexity and history of your systems you may have doubts that you were looking at all of the source or the latest version.

Looking into Database integrity may well throw up items like this screen shot. Where we have a number of different formats of email address and some extraneous text , similarly on the phone number list there is text and a variety of layouts. Finally we have the inevitable result of using ad-hoc fields with no validation or on screen guidance - transposed data mobile in email and vice versa.



Notes Field	Extra Info Field
(used for email address)	(used for mobile phone number)
jbloggs78@aol.com	cell: 7005896236
sherylc@msn.com	07568 456 321
HSMITH at SUN.COM	+447890987852
email: jbooth@mac.com	mobile 0754699888
07678 678912	suzip@btconnect.com

Hopefully this section raised awareness of the problems around changing and modernizing System i applications, the issues with finding out the necessary information and how seemingly straightforward issues can be time consuming and problematic. X-Analysis is designed and optimized to make the design recovery process as straightforward as possible as the rest of this concepts guide will illustrate.

Analysis, Documentation, & Application Subdivision

X-Analysis builds a very detailed repository over an entire application. The repository maintains all information about application objects, their relationships and all necessary information to obtain detailed information from each object across the entire system. Twenty years of ongoing development, over thousands of AS/400/iSeries/System-i applications written in all variants of RPGII/400/IV, COBOL, 2E and CL, has produced an unmatched capability to extract everything about an application from object right down to individual variables. The repository is built automatically using a single command, and initially collects all object related information, but then parses every source member in the specified system and every source line mapping the contextual information of each variable in the system. A certain amount of logical abstraction processing then takes place while building the repository to account for some of the idiosyncrasies typical in an RPG application. This includes constructs such as variable program calls, file overrides, prefixing and renaming in RPG. The repository thus represents a map of how the entire application functions right down to individual variables.

It's important to note that for 2E systems X-Analysis looks directly at the 2E model and derives its information from there i.e. not from the generated 2E code thus preserving the investment in the model.

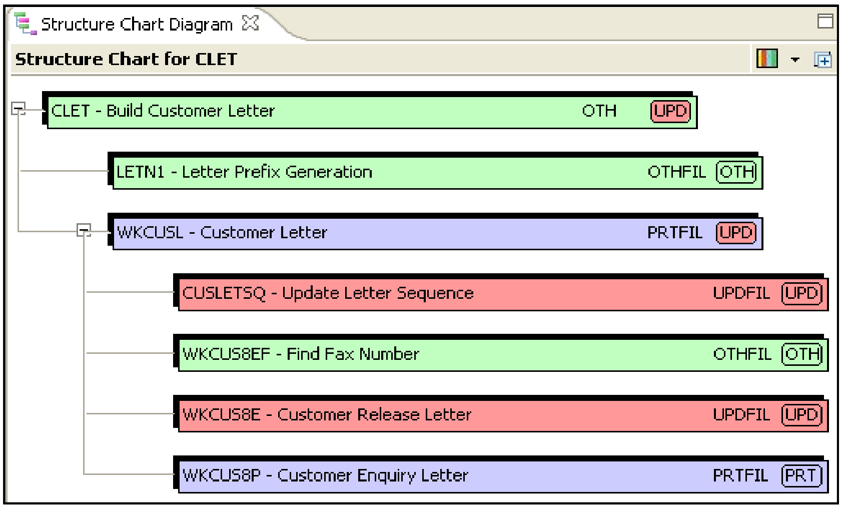
Understanding Design & Function More Easily

For efficient familiarization of an application's structure and general function, an abstraction above the source code combined with object-to-object relational information is required. A few simple but rich types of color-coded, graphical diagrams can reveal the data flow and architecture of individual objects or parts of an entire system. This is combined with automatically derived descriptions in the form of Pseudo narratives either in the diagrams or while browsing source code. The drill-down, go-anywhere-from-anywhere, interactive nature of these interfaces in the X-Analysis client provides a unique approach to information assimilation, allowing an analyst to gather information at high level or very detailed in an efficient and intuitive manner. The application abstraction is raised one level above implementation. This instantly removes complexity caused by the idiosyncrasies of different language versions and coding practices, typical in large legacy applications developed over many years.

Here is a brief description of some of these diagrammatic constructs and views:

Structure Chart Diagram - A Structure Chart Diagram (SCD) Display gives a graphic representation of how the control passes from one program to another program within the application. This follows the call structure down the complete stack. The diagram also reveals data input objects and also automatically derives a summarized description of each of the object in the diagram. Color-coding also reveals important functional aspects such as updates, prints, and displays, which help the user to zone in on commonly, sought after details.

Figure 1 - Structure Chart Diagram for a Program



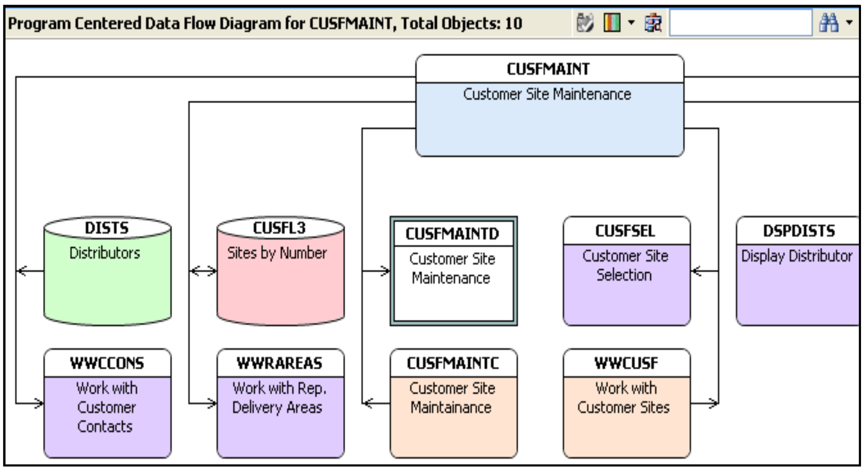
Structure Chart Diagram

Structure Chart for CLET

CLET - Build Customer Letter	OTH	UPD
LETN1 - Letter Prefix Generation	OTHFIL	OTH
WKCUSL - Customer Letter	PRTFIL	UPD
CUSLETSQ - Update Letter Sequence	UPDFIL	UPD
WKCUS8EF - Find Fax Number	OTHFIL	OTH
WKCUS8E - Customer Release Letter	UPDFIL	UPD
WKCUS8P - Customer Enquiry Letter	PRTFIL	PRT

Data Flow Diagram - A Data Flow Diagram (DFD) is a graphical representation of a program/object where used, showing the files and programs accessed by the subject object. It is also color-coded and shows both flow of data at a high object level, and contextual information about the specific variables/parameters passed between objects.

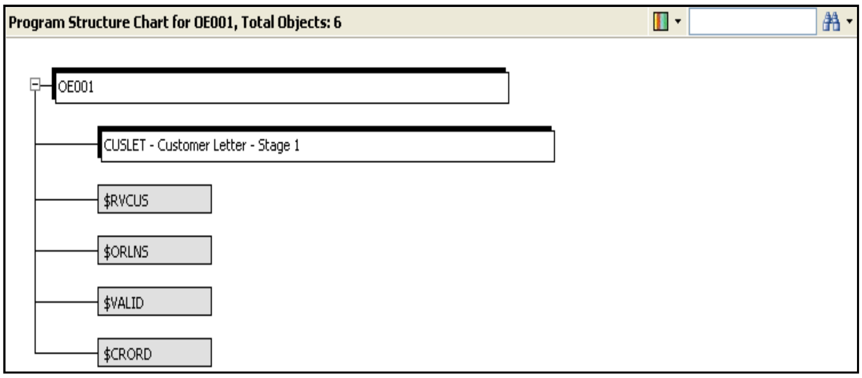
Figure 2 - Data Flow Diagram for a Program



Program Centered Data Flow Diagram for CUSFMAINT, Total Objects: 10

Program Structure Chart - A Program Structure Chart graphically displays the sequence of calls in the program. The call could be to execute a Subroutine / Program / Module / Service Program. For details, refer to X-Analysis User Manual.

Figure 3 - Program Structure Chart for a Program



Program Structure Chart for OE001, Total Objects: 6

OE001

CUSLET - Customer Letter - Stage 1

$RVCUS

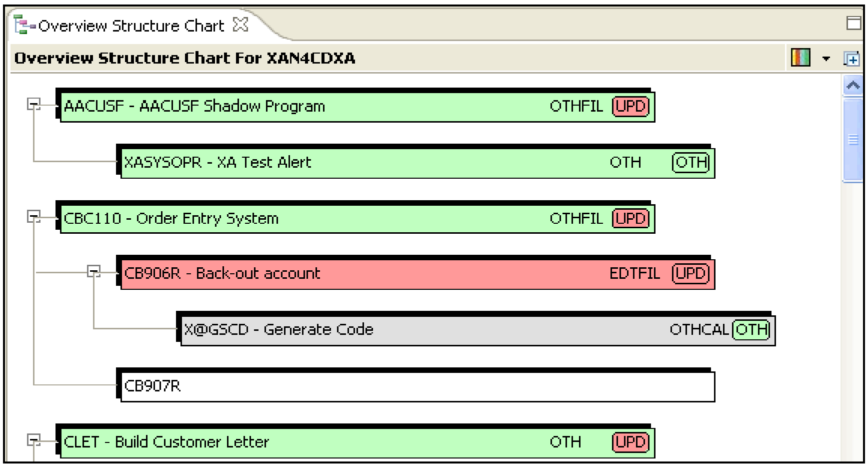
$ORLNS

$VALID

$CRORD

Overview Structure Chart - The Overview Structure Chart gives a snapshot of an application. It displays all the entry points to the application, and then the structure chart for each of these entry points.

Figure 4 - Overview Structure Chart for complete application



Overview Structure Chart

Overview Structure Chart For XAN4CDXA

AACUSF - AACUSF Shadow Program OTHFIL UPD

XASYSOPR - XA Test Alert OTH OTH

CBC110 - Order Entry System OTHFIL UPD

CB906R - Back-out account EDTFIL UPD

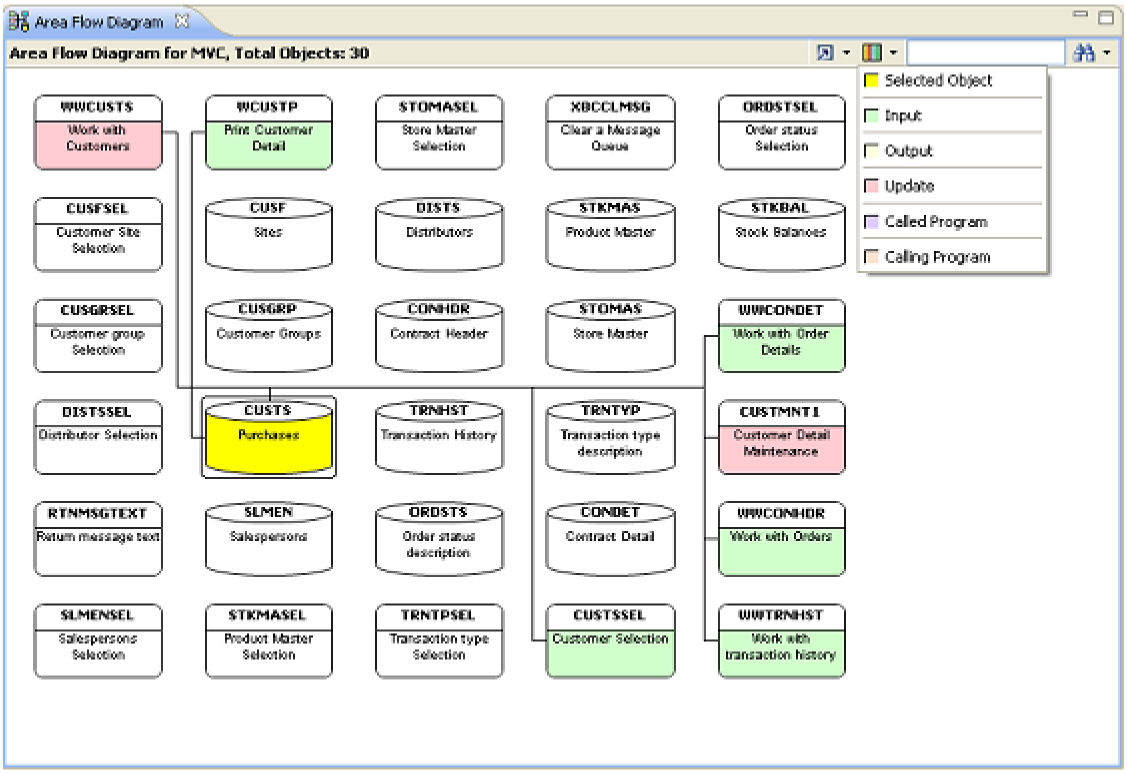
X@GSCD - Generate Code OTHCAL OTH

CB907R

CLET - Build Customer Letter OTH UPD

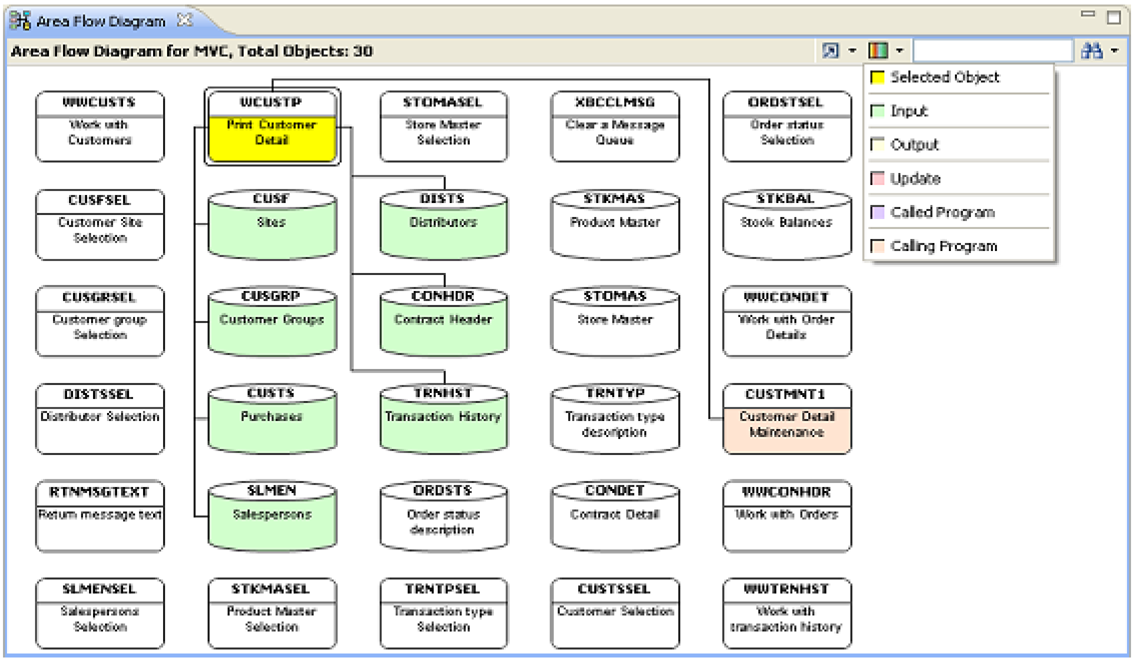
Area Flow Diagram - The Area Flow Diagram is a very useful interactive diagram that shows the linkages between files and programs within an application area, by clicking on a program we can see the files and programs it references , if we click on a file we see the programs that use it. The screen shot below shows the programs that use the Purchases file , the programs and files are colour coded to show whether they are Input/Output/Update or called or calling program.

Figure 5 – Area Flow diagram with File selected


Area Flow Diagram
Area Flow Diagram for MVC, Total Objects: 30
Selected Object
Input
Output
Update
Called Program
Calling Program
WWCUSTS Work with Customers
WCUSTP Print Customer Detail
STOMASEL Store Master Selection
XBCCLMSG Clear a Message Queue
ORDSTSEL Order status Selection
CUSFSEL Customer Site Selection
CUSF Sites
DISTS Distributors
STKMAS Product Master
STKBAL Stock Balances
CUSGRSEL Customer group Selection
CUSGRP Customer Groups
CONHDR Contract Header
STOMAS Store Master
WWCONDET Work with Order Details
DISTSSEL Distributor Selection
CUSTS Purchases
TRNHST Transaction History
TRNTYP Transaction type description
CUSTMNT1 Customer Detail Maintenance
RTNMSGTEXT Return message text
SLMEN Salespersons
ORDSTS Order status description
CONDET Contract Detail
WWCONHDR Work with Orders
SLMENSEL Salespersons Selection
STKMASEL Product Master Selection
TRNTPSEL Transaction type Selection
CUSTSSEL Customer Selection
WWTRNHST Work with transaction history

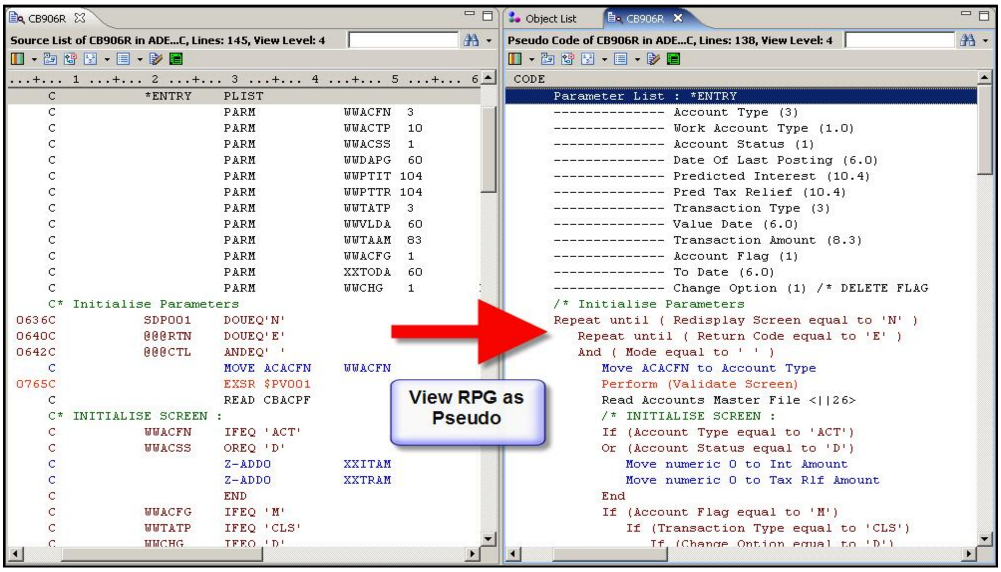
In this example we've clicked on a program and can see the files it uses as Input and also the program that calls it.

Figure 6 – Area Flow diagram with Program selected



RPG as Pseudo Code- With a single click, RPG can be viewed as a form of structured English or Pseudo code. Mnemonics' are substituted with file/field/variable texts and constants or literals.

Figure 7 - RPG to Pseudo Code with a Single Click

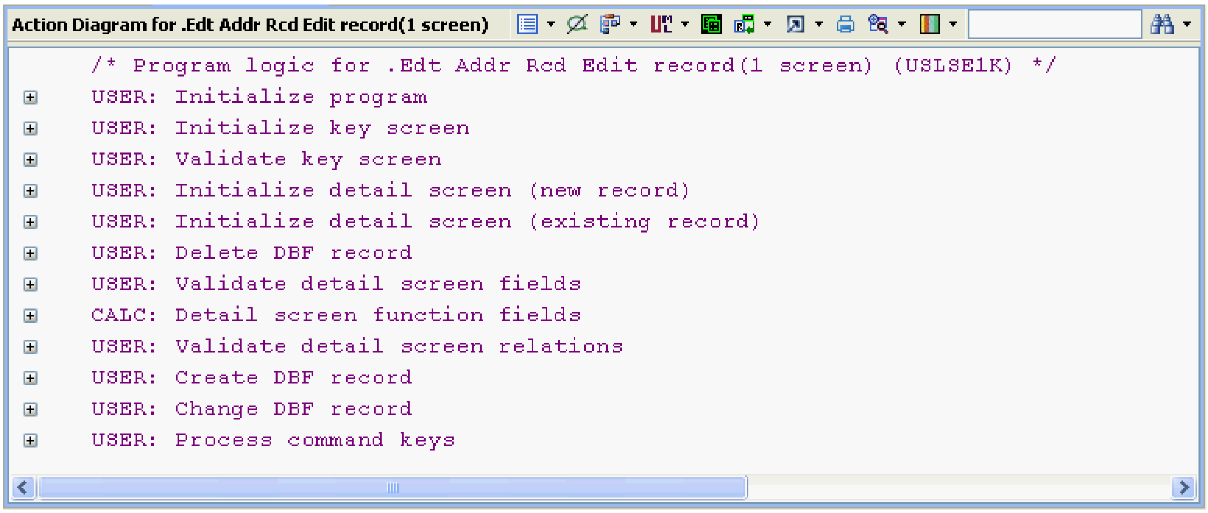

CB906R
Source List of CB906R in ADE...C, Lines: 145, View Level: 4
Object List
CB906R
Pseudo Code of CB906R in ADE...C, Lines: 138, View Level: 4
View RPG as Pseudo

```
...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6
    C            *ENTRY    PLIST
    C                      PARM           WWACFN  3
    C                      PARM           WWACTP  10
    C                      PARM           WWACSS  1
    C                      PARM           WWDAPG  60
    C                      PARM           WWPTIT 104
    C                      PARM           WWPTTR 104
    C                      PARM           WWTATP  3
    C                      PARM           WWVLDA  60
    C                      PARM           WWTAAM  83
    C                      PARM           WWACFG  1
    C                      PARM           XXTODA  60
    C                      PARM           WWCHG   1
    C* Initialise Parameters
0636C            SDP001    DOUEQ'N'
0640C            @@@RTN    DOUEQ'E'
0642C            @@@CTL    ANDEQ' '
    C                      MOVE ACACFN    WWACFN
0765C                      EXSR $PV001
    C                      READ CBACPF
    C* INITIALISE SCREEN :
    C            WWACFN    IFEQ 'ACT'
    C            WWACSS    OREQ 'D'
    C                      Z-ADDO         XXITAM
    C                      Z-ADDO         XXTRAM
    C                      END
    C            WWACFG    IFEQ 'M'
    C            WWTATP    IFEQ 'CLS'
    C            WWCHG     IFEQ 'D'
```

```
CODE
      Parameter List : *ENTRY
      -------------- Account Type (3)
      -------------- Work Account Type (1.0)
      -------------- Account Status (1)
      -------------- Date Of Last Posting (6.0)
      -------------- Predicted Interest (10.4)
      -------------- Pred Tax Relief (10.4)
      -------------- Transaction Type (3)
      -------------- Value Date (6.0)
      -------------- Transaction Amount (8.3)
      -------------- Account Flag (1)
      -------------- To Date (6.0)
      -------------- Change Option (1) /* DELETE FLAG
      /* Initialise Parameters
      Repeat until ( Redisplay Screen equal to 'N' )
         Repeat until ( Return Code equal to 'E' )
         And ( Mode equal to ' ' )
            Move ACACFN to Account Type
            Perform (Validate Screen)
            Read Accounts Master File <||26>
            /* INITIALISE SCREEN :
            If (Account Type equal to 'ACT')
            Or (Account Status equal to 'D')
               Move numeric 0 to Int Amount
               Move numeric 0 to Tax Rlf Amount
            End
            If (Account Flag equal to 'M')
               If (Transaction Type equal to 'CLS')
                  If (Change Option equal to 'D')
```

2E as Pseudo Code- 2E action diagrams can also be displayed as pseudo-code. The information returned contains both the 2E variable names and contexts alongside the RPG/COBOL mnemonic.

Figure 8 – 2E Action Diagram



```
/* Program logic for .Edt Addr Rcd Edit record(1 screen) (USLSE1K) */
USER: Initialize program
USER: Initialize key screen
USER: Validate key screen
USER: Initialize detail screen (new record)
USER: Initialize detail screen (existing record)
USER: Delete DBF record
USER: Validate detail screen fields
CALC: Detail screen function fields
USER: Validate detail screen relations
USER: Create DBF record
USER: Change DBF record
USER: Process command keys
```

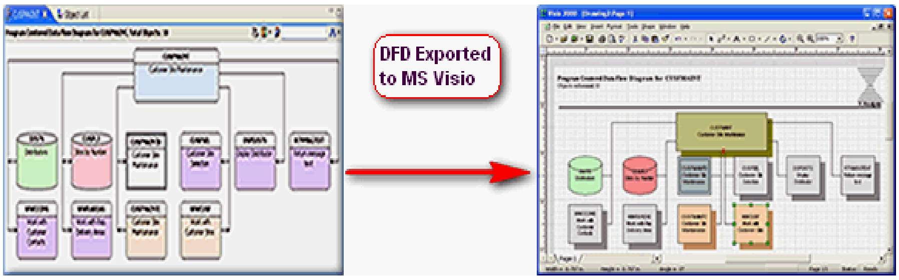
There is a section on 'Working with Action Diagram' later in the guide.

Producing Static Documentation Automatically

Interactive analysis via a graphical client is generally the most intuitive manner in which to analyze a system, but there is often a requirement for various types of static information in the form of structured documentation. Examples of this are project documentation, auditing information, testing instructions, and customer support documentation (such as with ISV supplied business software). X-Analysis produces a number of these outputs including:

Data Flow Chart in MS Visio - Any interactive diagram produced by X-Analysis 8 in the client, can be automatically exported to MS Visio. . In addition to this, an RPG/COBOL program or 2E action diagram can be produced as a data flow chart interactively while browsing the source from within X-Analysis. If the RPG program is in Pseudo Code mode, the Data Flow Chart will use the narratives from the Pseudo code. This enables non-system i technologists and analysts to assimilate information at a detailed level of the application without any dependency on RPG, COBOL or 2E experts.

Figure 9- DFD Exported to MS Visio



Lists and Results sets – Any source, object, or impact-analysis result list can be directly exported to formatted MS Excel or Word from the client.

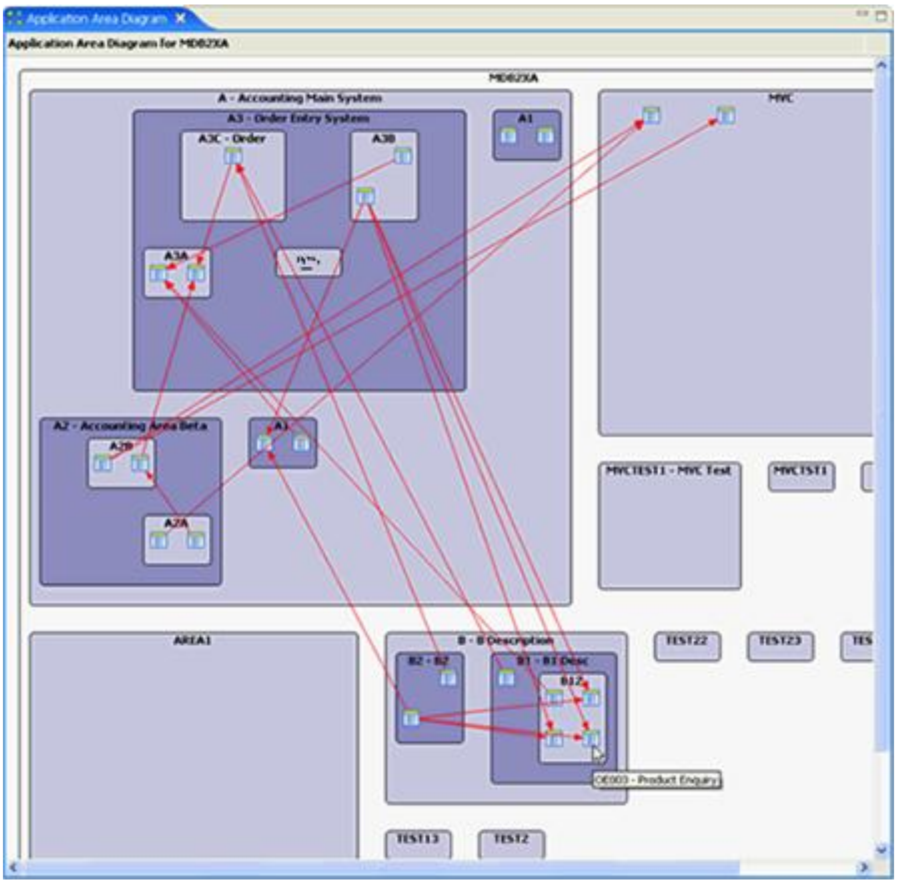
MS Word Project Documentation Wizard – With the use of a simple wizard, documents that might take weeks to produce manually, allow the user to select any of the graphical diagrams, lists, flowcharts, annotation and business rules summaries generated interactively by the client interface, can be collated into a single document with contents and index. This can be done for a single object, an application area (explained in the next section), a list of objects, or an entire system. Any of these documents can then be edited and distributed as required.

Dividing Systems into Application Areas

Entire legacy applications are often too large to effectively comprehend or effect wholesale change. For this reason it is often necessary or helpful to sub-divide a system into application areas. The reasons and specifications for these may change with time too. X-Analysis provides facilities for subdividing an application area into groups of objects that meet user defined selection criteria. These criteria might be based on function or even generic name. X-Analysis then uses the sophisticated cross-reference information and Data Model relationships to include, automatically all related elements such as programs, displays, or files in the application area.

Application areas filters can then be used through the X-Analysis Solution Sets to view, document or reengineer as opposed to individual objects.

Figure 10 - Application Area Diagram for System Overviews



The Application Area diagram in X-Analysis is interactive and by clicking on different parts of your system you can see the relationships between either all parts or just the area you've clicked on and the areas it relates to.

Recovering an Application Design

The concept of reusing existing code or logic is not a new one. The challenge has always been to identify, isolate, and reuse only those designs that are relevant in the new context in which they are desirable. The sheer volume of code, its complexity, and the general lack of resources to understand legacy languages, specifically RPG, represents a tragic potential waste of valuable business assets. In many cases, these expensive and well-established legacy designs have little chance of even having their relevance assessed, let alone being reused. The Design Recovery Solution Set of X-Analysis addresses this problem more directly, by isolating, indexing, and documenting those design elements that could be relevant in a modern version of the application being assessed.

UML Diagramming

The Design recovery module of X-Analysis translates the recovered information as UML diagrams. The objective of UML diagrams in this context is to help sketch application designs and to make such sketches portable and reusable in other IDE's such as Rational, Borland, MyEclipse, etc. The three diagrams automatically generated by X-Analysis are:

Activity Diagram – Activity diagrams illustrate the dynamic nature of a system by modeling the flow of control from activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system. Typically, activity diagrams are used to model workflow or business processes and internal operation. X-Analysis produces these automatically either from a single program with multiple screens, or a group of programs. Each activity in the diagram represents a usable screen format in the RPG program. A user can also view the extracted Business Rules, relevant to that particular activity/format directly from within the diagram.

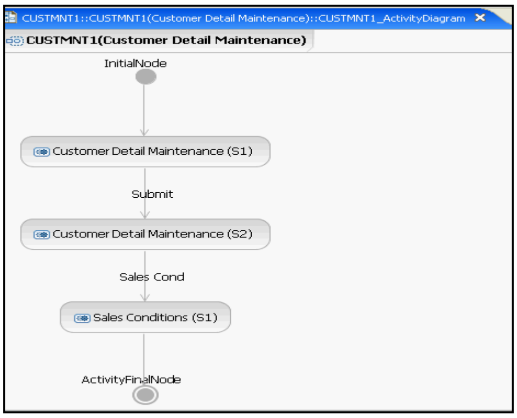

CUSTMNT1::CUSTMNT1(Customer Detail Maintenance)::CUSTMNT1_ActivityDiagram
CUSTMNT1(Customer Detail Maintenance)
InitialNode
Customer Detail Maintenance (S1)
Submit
Customer Detail Maintenance (S2)
Sales Cond
Sales Conditions (S1)
ActivityFinalNode

Figure 11 – Activity Diagram for a Program

The guide uses Activity diagram as the beacon to understand and review the artifacts recovered in the Design recovery process.

Use Case Diagram – Use Case Diagrams model the functionality of system using actors and use cases. Use cases are services or functions provided by the system to its users. Auto-generated from X-Analysis, this can be used as an alternative view to the Activity Diagram, and also has drill-down capabilities for viewing extracted Business Rules.

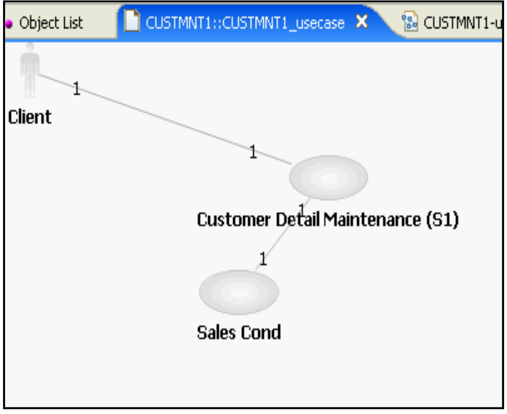
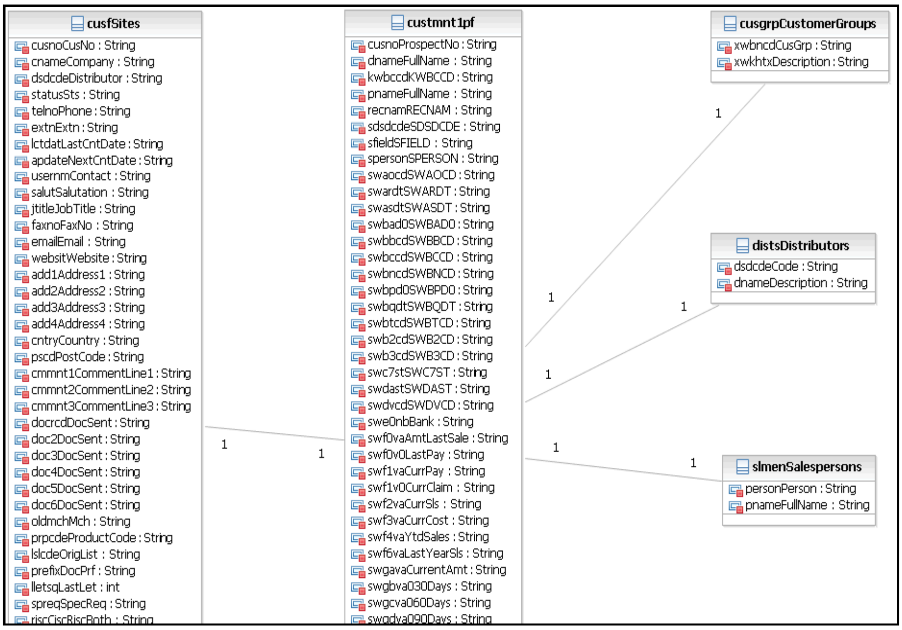


Figure 12 – Use Case Diagram for a Program

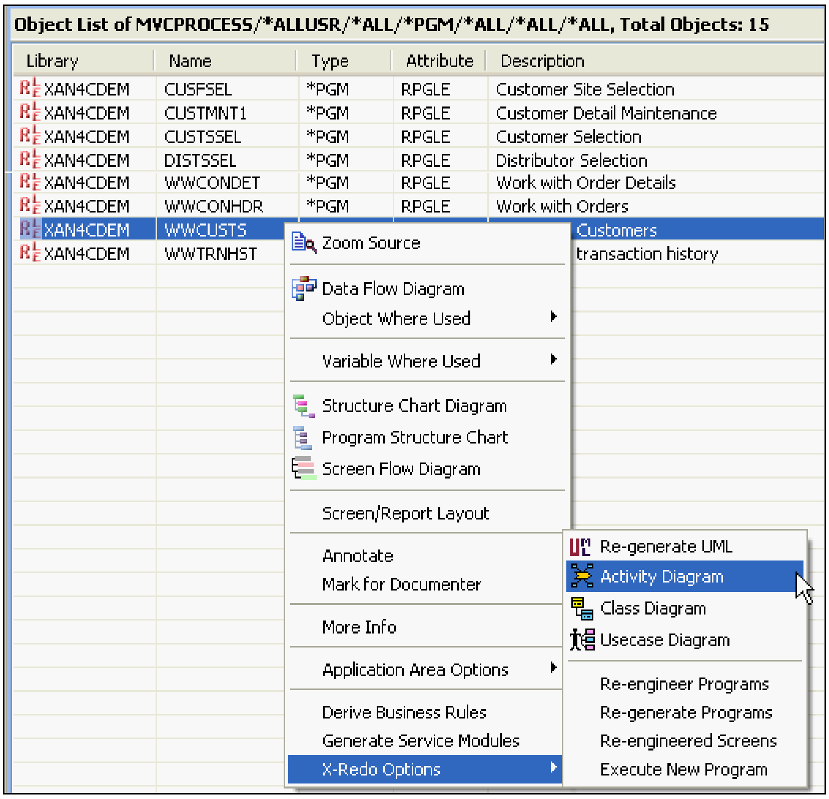
Class Diagram – Class diagrams are the backbone of all object-oriented methods, including UML. They describe the static structure of a system. Classes represent an abstraction of entities with common characteristics. Associations represent the relationships between classes. An extracted class in a class diagram corresponds to the individual screen formats and all of the specific attributes of that particular format. X-Analysis deduces the links between these classes using a combination of the derived data model, and call or action information extracted from each program.

Figure 13 – Class Diagram for a Program



Producing any of these diagrams from within X-Analysis is as simple as right-clicking on an object and selecting the appropriate option from the menu:

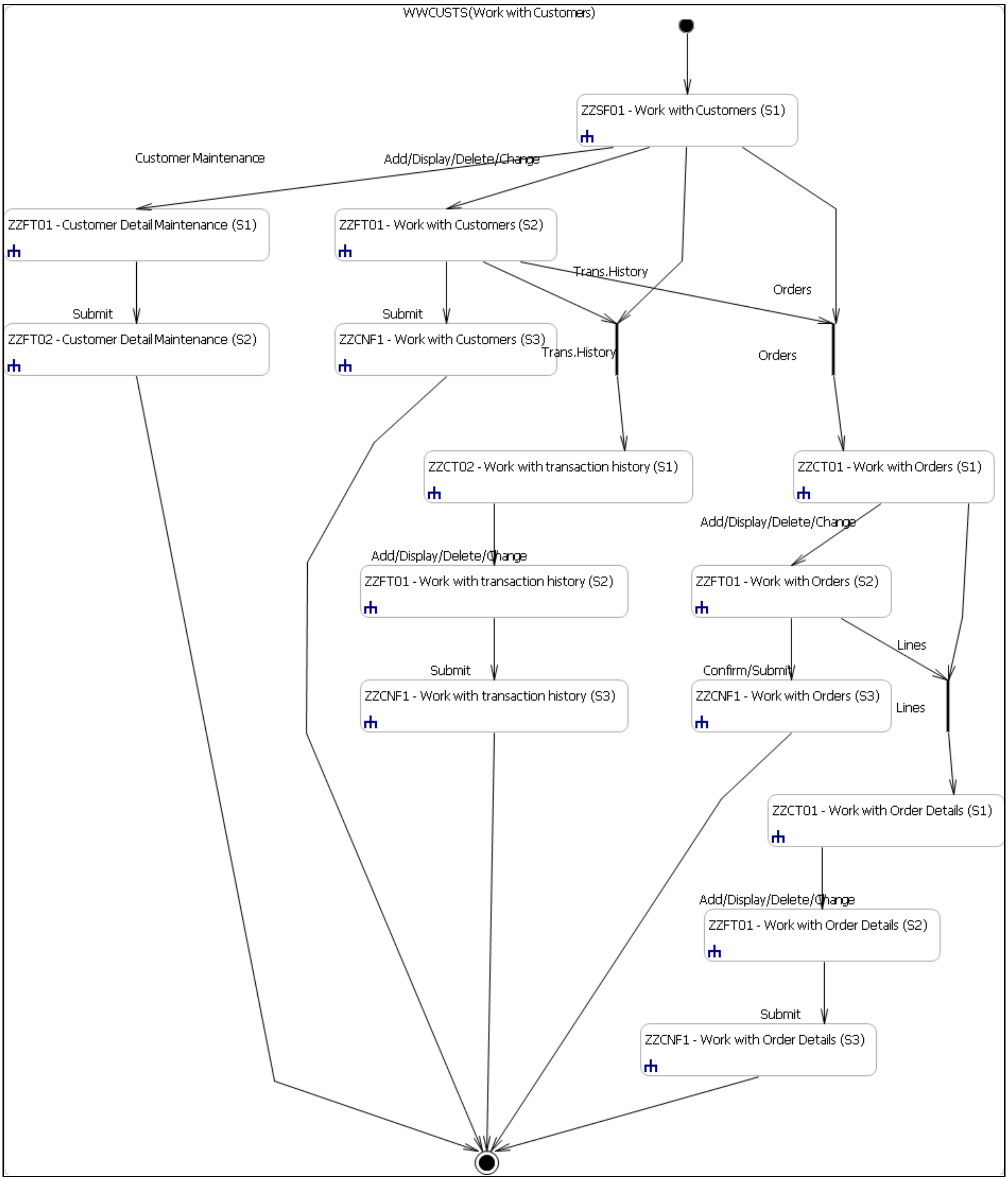
Figure 14 – Producing UML Diagrams in X-Analysis



Object List of MYCPROCESS/*ALLUSR/*ALL/*PGM/*ALL/*ALL/*ALL, Total Objects: 15

Library	Name	Type	Attribute	Description
XAN4CDEM	CUSFSEL	*PGM	RPGLE	Customer Site Selection
XAN4CDEM	CUSTMNT1	*PGM	RPGLE	Customer Detail Maintenance
XAN4CDEM	CUSTSSEL	*PGM	RPGLE	Customer Selection
XAN4CDEM	DISTSSEL	*PGM	RPGLE	Distributor Selection
XAN4CDEM	WWCONDET	*PGM	RPGLE	Work with Order Details
XAN4CDEM	WWCONHDR	*PGM	RPGLE	Work with Orders
XAN4CDEM	WWCUSTS			Customers
XAN4CDEM	WWTRNHST			transaction history

Figure 15 – Activity Diagram


WWCUSTS(Work with Customers)
ZZSF01 - Work with Customers (S1)
Customer Maintenance
Add/Display/Delete/Change
ZZFT01 - Customer Detail Maintenance (S1)
ZZFT01 - Work with Customers (S2)
Trans.History
Orders
Submit
Submit
ZZFT02 - Customer Detail Maintenance (S2)
ZZCNF1 - Work with Customers (S3)
Trans.History
Orders
ZZCT02 - Work with transaction history (S1)
ZZCT01 - Work with Orders (S1)
Add/Display/Delete/Change
Add/Display/Delete/Change
ZZFT01 - Work with transaction history (S2)
ZZFT01 - Work with Orders (S2)
Lines
Submit
Confirm/Submit
ZZCNF1 - Work with transaction history (S3)
ZZCNF1 - Work with Orders (S3)
Lines
ZZCT01 - Work with Order Details (S1)
Add/Display/Delete/Change
ZZFT01 - Work with Order Details (S2)
Submit
ZZCNF1 - Work with Order Details (S3)

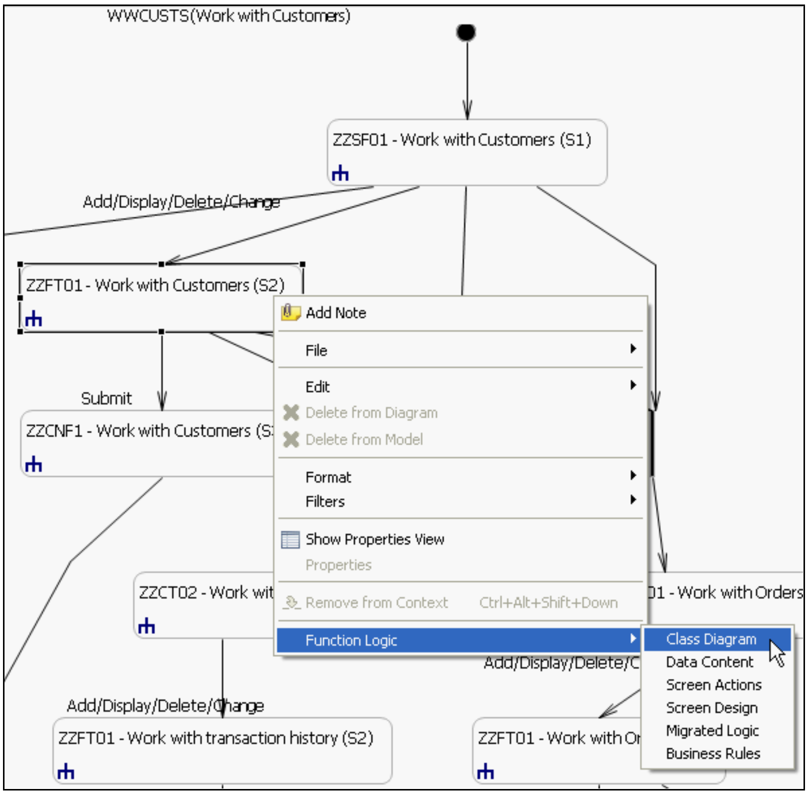
Working with UML Activity diagram

Activity Diagram on a program comes with host of options to explore the design elements recovered in the Design recovery process. The user can analyze the following elements of a program from its Activity diagram:

- Class Diagram
- Data Content
- Screen Actions
- Screen Design
- Migrated Logic
- Business Rules

The screen below shows how each node on the Activity diagram can be further explored for recovered artifacts.

Figure 16 – Options on Activity Diagram node



Data Content

The relational model of an enterprise application is an extremely powerful piece of information and potentially valuable asset to the organization. Unlike 2E systems for almost all RPG or COBOL applications running on System i, there is no explicit data model or schema defined. By the term model, we are referring to the foreign key or relational model, not just the physical model of the database. The relational model or architecture of the database can be reused in a number of scenarios including:

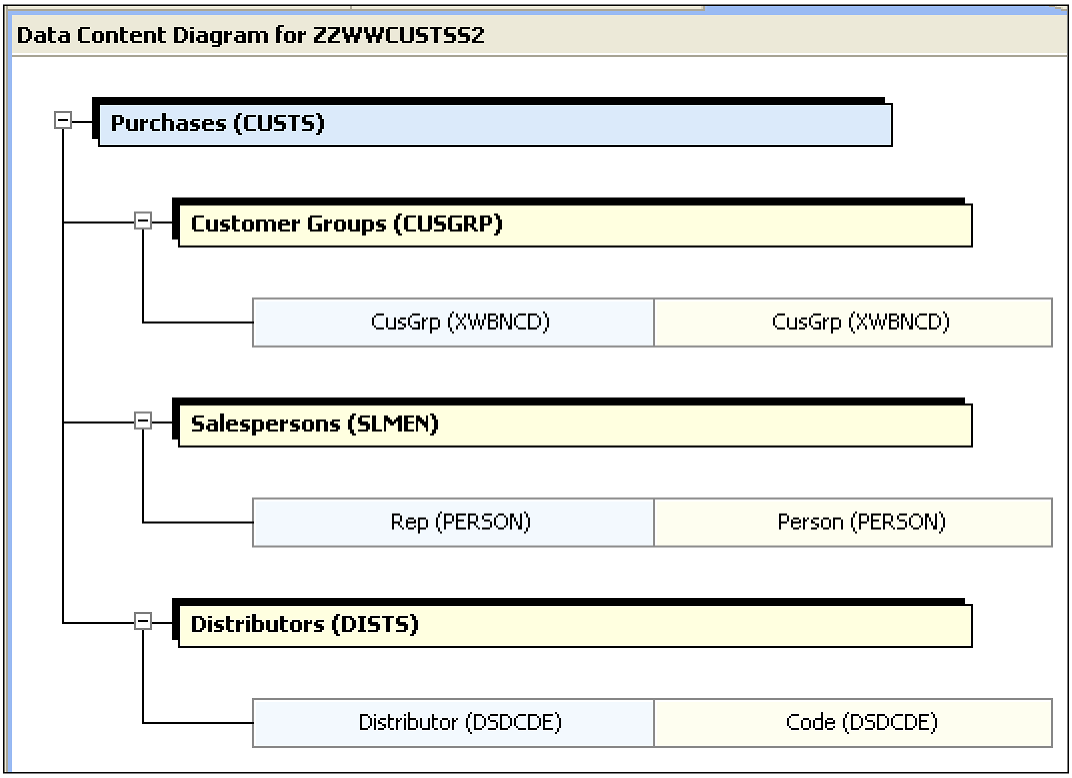
- Understanding application architecture
- Data quality analysis – referential integrity testing
- Automated test data extraction, scrambling and aging
- Building BI applications or Data warehouses

X-Analysis has the unique capability of automatically deriving the explicit system data model from a legacy RPG, COBOL or 2E application. Let us have a look at this and the model reuse capability in a bit more detail.

Deriving the Legacy Data Model – X-Analysis accomplishes this by analyzing the data structures of the physical and logical files, but it then programmatically traces these through all programs that use them to verify the existence of any cross-file relationships or foreign keys. These derived relationships can also be verified by the product by performing an integrity check on the actual data. This ensures that the data of the dependent file makes a reference, to data records from the owning file. In this way, the automated reverse engineering can fully extract the data model from even the most complex legacy system.

Data Content diagram generated for a node on the Activity diagram uses the model information to depict the primary file in use by that node (or Screen) shown in blue below. The sub nodes are the secondary files (in light yellow) with which the primary file joins to pull the additional information to show on the screen. The third node is split into two parts. The left section, in blue, lists the field(s) from the primary file and the right section, in light yellow, lists the field(s) from the secondary file facilitating the file join.

Figure 17 – Data Content Diagram

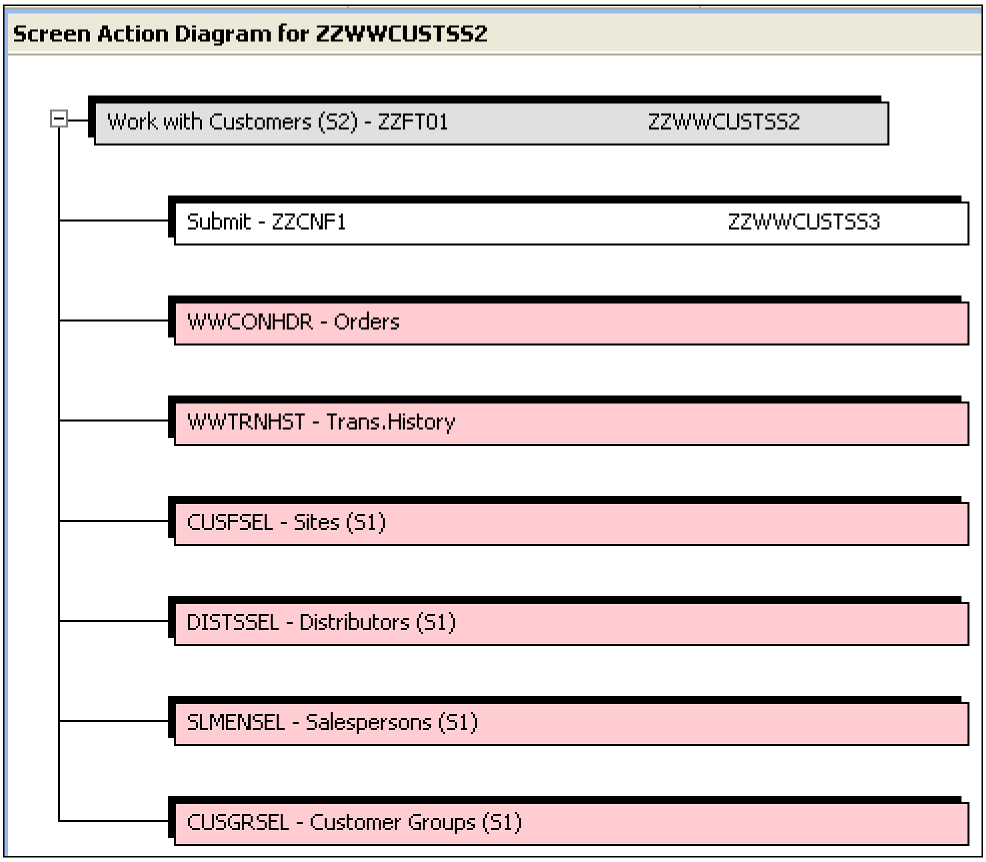


Screen Actions

Screen Action diagram presents the actions allowed on the node/screen. The actions could be:

- Prompt Action – To look up for the possible values for a field, facilitated by the join on the secondary/foreign file.
- Submit Action – Usually on the Record screen or a Form, which submits the data for validation and moving on.
- Navigation Action – To branch off to a new screen. Usually on the Grids or Record screens being viewed in display mode.

Figure 18 – Screen Action Diagram



Screen Action Diagram for ZZWWCUSTSS2

Work with Customers (S2) - ZZFT01 ZZWWCUSTSS2

Submit - ZZCNF1 ZZWWCUSTSS3

WWCONHDR - Orders

WWTRNHST - Trans.History

CUSFSEL - Sites (S1)

DISTSSEL - Distributors (S1)

SLMENSEL - Salespersons (S1)

CUSGRSEL - Customer Groups (S1)

Screen Design

The screens of a legacy application are a classic example where the design is useful in a modernization context, and the code is not. All modern IDE's provide powerful UI development tools. Modern UI standards and preferences for style and technology also vary from project to project. The sheer number of screens in a legacy application presents a logistical problem in recreating them manually, even with the cleverest developers and best tooling. X-Analysis lets you see what the legacy screen looked like without having to run the application which is a great time saver for people who haven't been involved with the original application:

Screen designs of legacy applications are not just about look and feel, there are attributes, and logic embedded which from a design point of view is relevant, no matter what technology being used to implement them. These are:

Formats/Layouts – Some screens may benefit from amalgamation or redesign, but table edits, and non-transaction type screens will largely remain the same, if not identical in layout.

Actions – Whether from sub-file options, command keys, or default enter actions, these often represent an important part of the usefulness of an application design. The mechanisms used to offer or invokes these calls may change, but where they go logically and what parameters they pass will largely remain consistent.

Fields/Files/Attributes – What fields are on what screens, and where the data comes from is a requirement in any system development. Attributes of a field can also help determine what type of controls might be used in a modern UI. For example, a Boolean type might be implemented with a check box, a date with a date prompt. Again, these are simple enough to edit in modern IDE's, but the volume associated with any large legacy application modernization can make this work prohibitive.

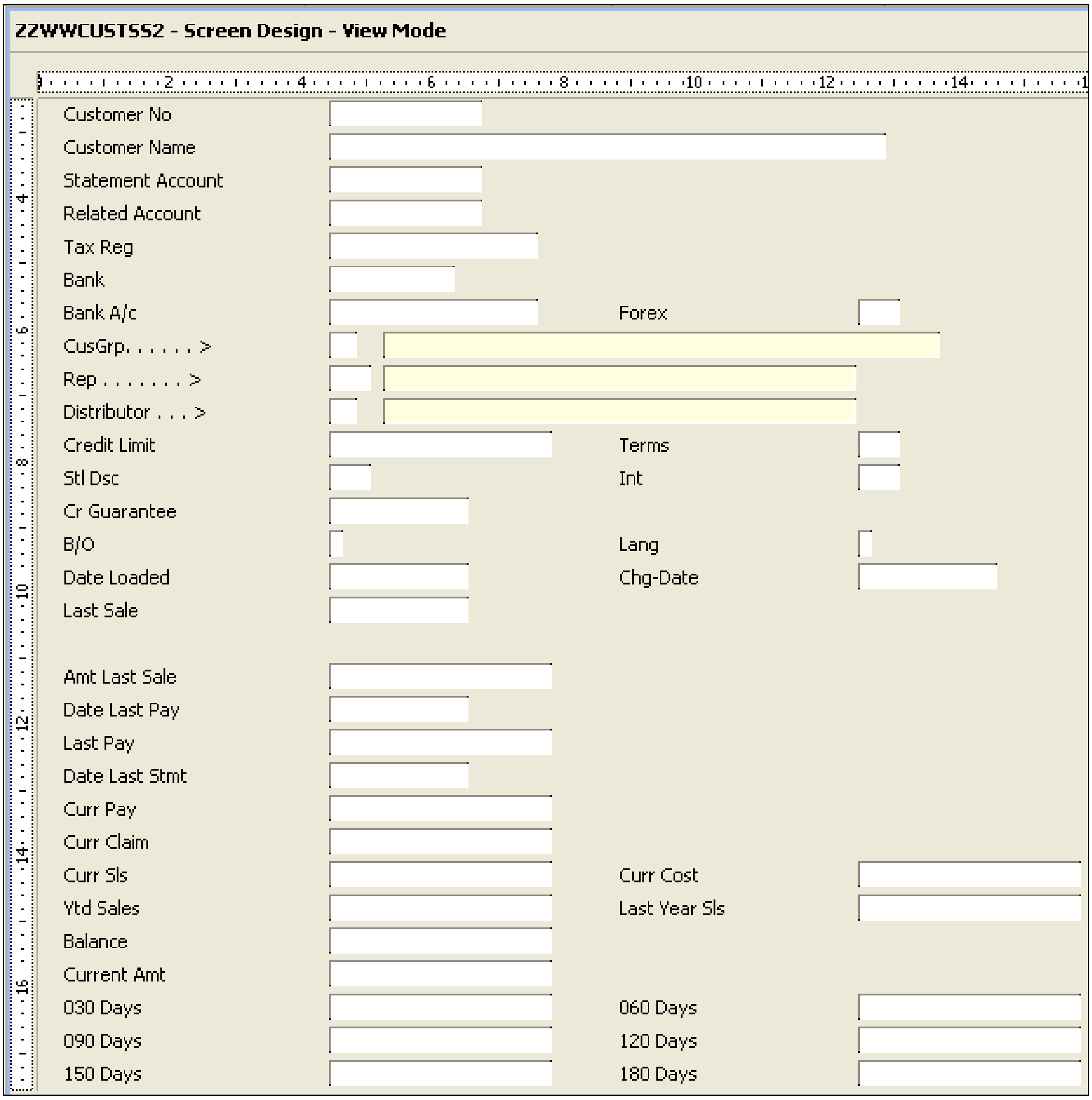
Data Model Mapping – Validations and prompting mechanisms that ensure referential integrity in legacy applications can also be vital to extract. This is both to implement referential integrity and to provide design information for building modern prompt or selection controls such as drop downs or lists.

Naturally, it will be desirable to redesign some UI's completely. For those programs and screens where this is not the case, the design, and mapping information can be used directly in the new version of the application, even though the UI code has been discarded.

X-Analysis extracts User Interface design information as described above and stores it as meta-data in the X-Analysis repository. This is used as reference documentation for rebuilding UI's manually, or for programmatically regenerating new View and Controller artifacts in the chosen new technology. The design meta-data can also naturally be used to generate new interfaces using any technology such as EGL, Ajax, RCP, C#, VB or even RPG.

The Screen Design presents the recovered screen layout.

Figure 19 – Screen Design



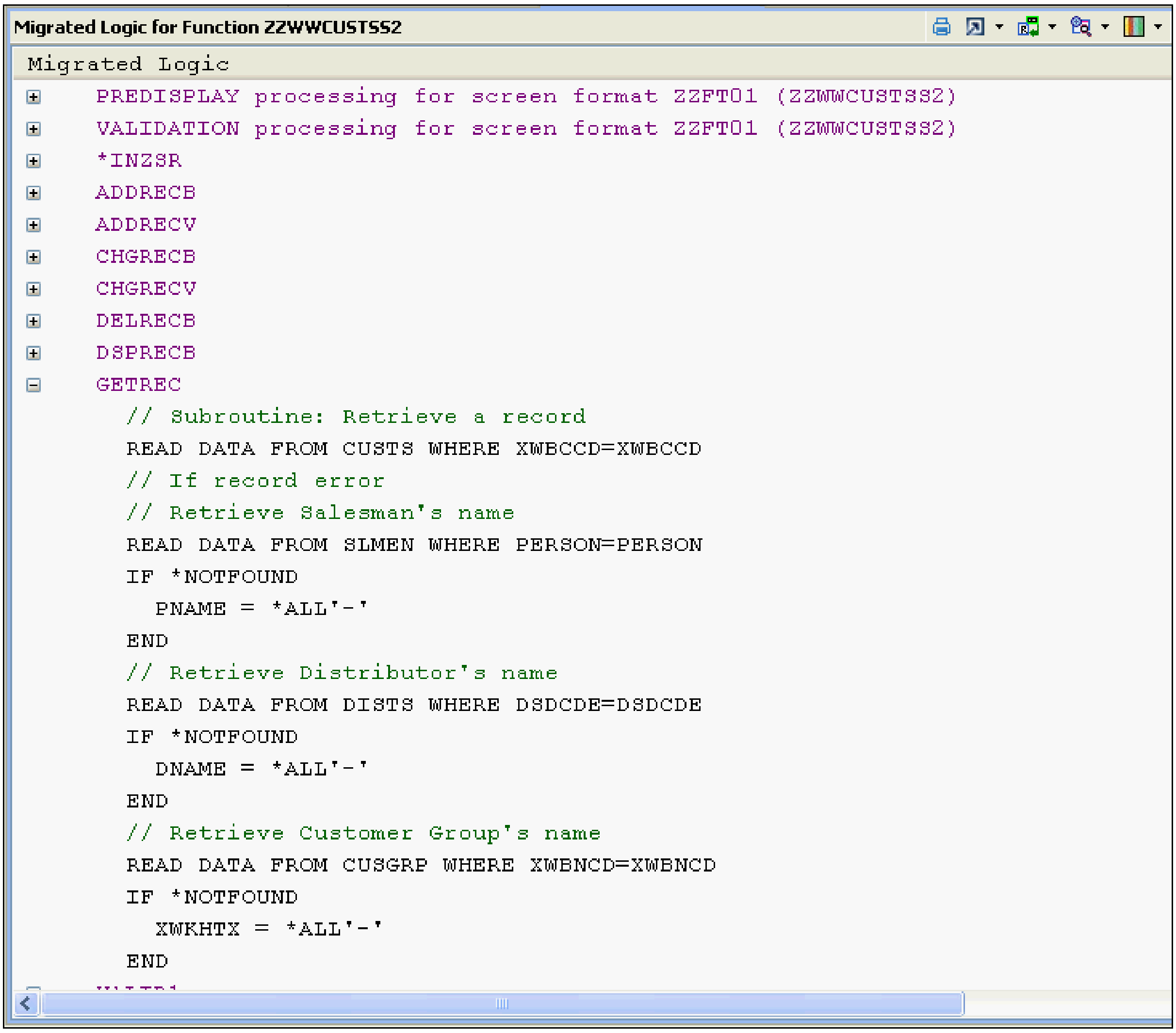
ZZWWCUSTSS2 - Screen Design - View Mode

Customer No	
Customer Name	
Statement Account	
Related Account	
Tax Reg	
Bank	
Bank A/c	Forex
CusGrp. . . . . . >	
Rep . . . . >	
Distributor . . . >	
Credit Limit	Terms
Stl Dsc	Int
Cr Guarantee	
B/O	Lang
Date Loaded	Chg-Date
Last Sale	
Amt Last Sale	
Date Last Pay	
Last Pay	
Date Last Stmt	
Curr Pay	
Curr Claim	
Curr Sls	Curr Cost
Ytd Sales	Last Year Sls
Balance	
Current Amt	
030 Days	060 Days
090 Days	120 Days
150 Days	180 Days

Migrated Logic

The design recovery process extracts the business logic from the legacy application and holds it as simple 'English like' code in the repository. This is an intermediate state of the code recovered and is termed Migrated Logic. The Migrated Logic for the Re-engineered screen presents the screen logic in a simple and more readable format. This is used by the Redevelopment module to rebuild the legacy application in the desired language e.g. Java/EGL/.Net/PHP.

Figure 20 – Migrated Logic for a program



```
⊞    PREDISPLAY processing for screen format ZZFT01 (ZZWWCUSTSS2)
⊞    VALIDATION processing for screen format ZZFT01 (ZZWWCUSTSS2)
⊞    *INZSR
⊞    ADDRECB
⊞    ADDRECV
⊞    CHGRECB
⊞    CHGRECV
⊞    DELRECB
⊞    DSPRECB
⊟    GETREC
       // Subroutine: Retrieve a record
       READ DATA FROM CUSTS WHERE XWBCCD=XWBCCD
       // If record error
       // Retrieve Salesman's name
       READ DATA FROM SLMEN WHERE PERSON=PERSON
       IF *NOTFOUND
         PNAME = *ALL'-'
       END
       // Retrieve Distributor's name
       READ DATA FROM DISTS WHERE DSDCDE=DSDCDE
       IF *NOTFOUND
         DNAME = *ALL'-'
       END
       // Retrieve Customer Group's name
       READ DATA FROM CUSGRP WHERE XWBNCD=XWBNCD
       IF *NOTFOUND
         XWKHTX = *ALL'-'
       END
```

Business Rules

Once the system UI, data access & data model has been recovered & the application has been rebuilt or rewritten from this design, it is then necessary to extract the logic that gives the application its particular characteristics. The generic term for such logic is Business Rules. The challenge is to extract or “harvest” these rules from the legacy code.

Traditionally Business analysts or consultants find the rules for a new application by organizing workshops and interviews then manually writing use cases to describe the rules as text. However, for a legacy application all the rules are already there prescribed in the application code - you just have be able to retrieve it.

The problem is that in the vast majority of legacy RPG and COBOL programs, the business rule logic is mixed in with screen handling, database I/O, and flow control. So harvesting these business rules from legacy applications requires knowledge of the application and the language used to implement it, both of which are steadily diminishing resource. Once harvested these rules need to be narrated and indexed, thus providing critical information for any analysts, architect or developer charged with rebuilding a legacy application. The task of harvesting business rules is therefore a highly skilled, labor-intensive, and costly exercise for any organization.

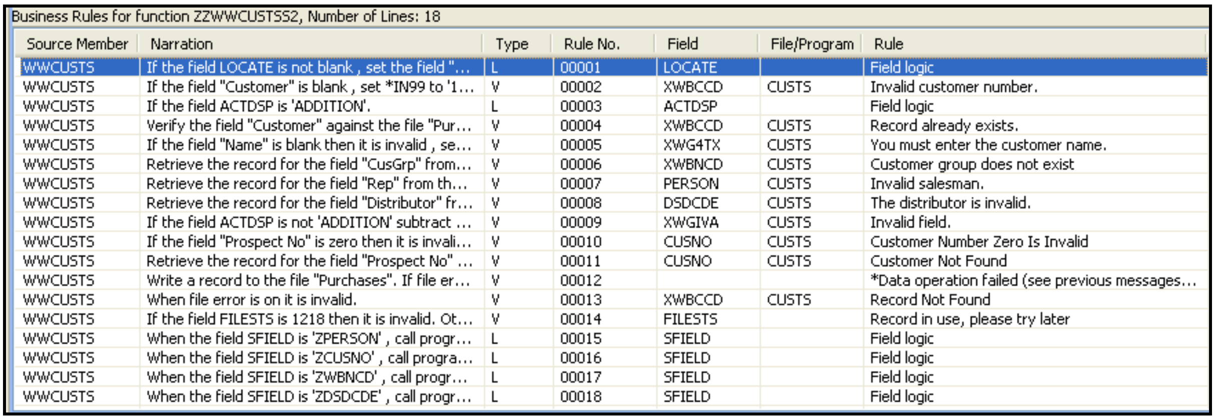
X-Analysis accomplishes this task by automatically scanning the RPG and COBOL programs and 2E model programmatically. It then separates out rule code from the body of the application and identifies, indexes, narrates, and stores business rule logic code into a structured, usable repository. In the final part of the process, it supplies appropriate text narratives to describe these harvested rules.

Business rules are defined as discreet blocks of program logic, gathered during the design recovery process, which describe data processes that are independent of both the data model and the environment. X-Analysis identifies the following business rule types:

- C = Call to a batch program
- E = Call to a validation routine
- L = Field logic
- P = Setting an entry parameter (Batch programs only)
- Q = Setting a parameter for a called batch program
- R = Non-owner file read
- U = Secondary update
- V = Validation
- W = Setting a screen work field
- X = Synon user exit

The rule narration too is held along with the rule for ease of understanding.

Figure 21 – Business Rules for a program



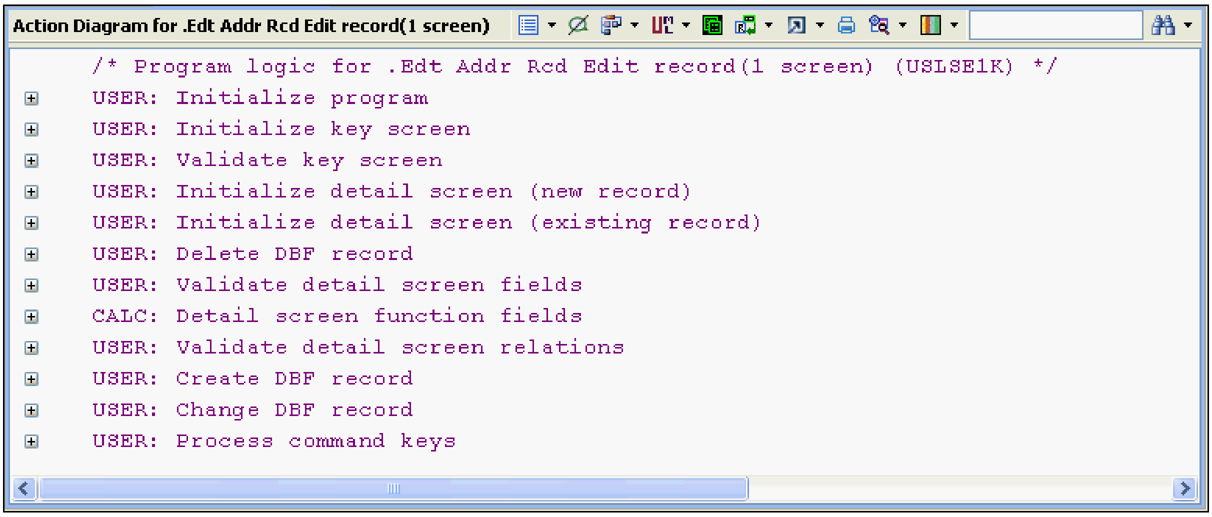
Business Rules for function ZZWWCUSTSS2, Number of Lines: 18

Source Member	Narration	Type	Rule No.	Field	File/Program	Rule
WWCUSTS	If the field LOCATE is not blank , set the field "…	L	00001	LOCATE		Field logic
WWCUSTS	If the field "Customer" is blank , set *IN99 to '1…	V	00002	XWBCCD	CUSTS	Invalid customer number.
WWCUSTS	If the field ACTDSP is 'ADDITION'.	L	00003	ACTDSP		Field logic
WWCUSTS	Verify the field "Customer" against the file "Pur…	V	00004	XWBCCD	CUSTS	Record already exists.
WWCUSTS	If the field "Name" is blank then it is invalid , se…	V	00005	XWG4TX	CUSTS	You must enter the customer name.
WWCUSTS	Retrieve the record for the field "CusGrp" from…	V	00006	XWBNCD	CUSTS	Customer group does not exist
WWCUSTS	Retrieve the record for the field "Rep" from th…	V	00007	PERSON	CUSTS	Invalid salesman.
WWCUSTS	Retrieve the record for the field "Distributor" fr…	V	00008	DSDCDE	CUSTS	The distributor is invalid.
WWCUSTS	If the field ACTDSP is not 'ADDITION' subtract …	V	00009	XWGIVA	CUSTS	Invalid field.
WWCUSTS	If the field "Prospect No" is zero then it is invali…	V	00010	CUSNO	CUSTS	Customer Number Zero Is Invalid
WWCUSTS	Retrieve the record for the field "Prospect No" …	V	00011	CUSNO	CUSTS	Customer Not Found
WWCUSTS	Write a record to the file "Purchases". If file er…	V	00012			*Data operation failed (see previous messages…
WWCUSTS	When file error is on it is invalid.	V	00013	XWBCCD	CUSTS	Record Not Found
WWCUSTS	If the field FILESTS is 1218 then it is invalid. Ot…	V	00014	FILESTS		Record in use, please try later
WWCUSTS	When the field SFIELD is 'ZPERSON' , call progr…	L	00015	SFIELD		Field logic
WWCUSTS	When the field SFIELD is 'ZCUSNO' , call progra…	L	00016	SFIELD		Field logic
WWCUSTS	When the field SFIELD is 'ZWBNCD' , call progr…	L	00017	SFIELD		Field logic
WWCUSTS	When the field SFIELD is 'ZDSDCDE' , call progr…	L	00018	SFIELD		Field logic

Working with 2E Action Diagram

X-Analysis nicely extracts the Action Diagrams detailing the Business logic the application contains. Zooming into an Action Diagram is just a mouse click away. On X-Analysis Program's object list, double-click on a program and you are presented with the Action Diagram (shown below).

Figure 22 – Action Diagram for a program



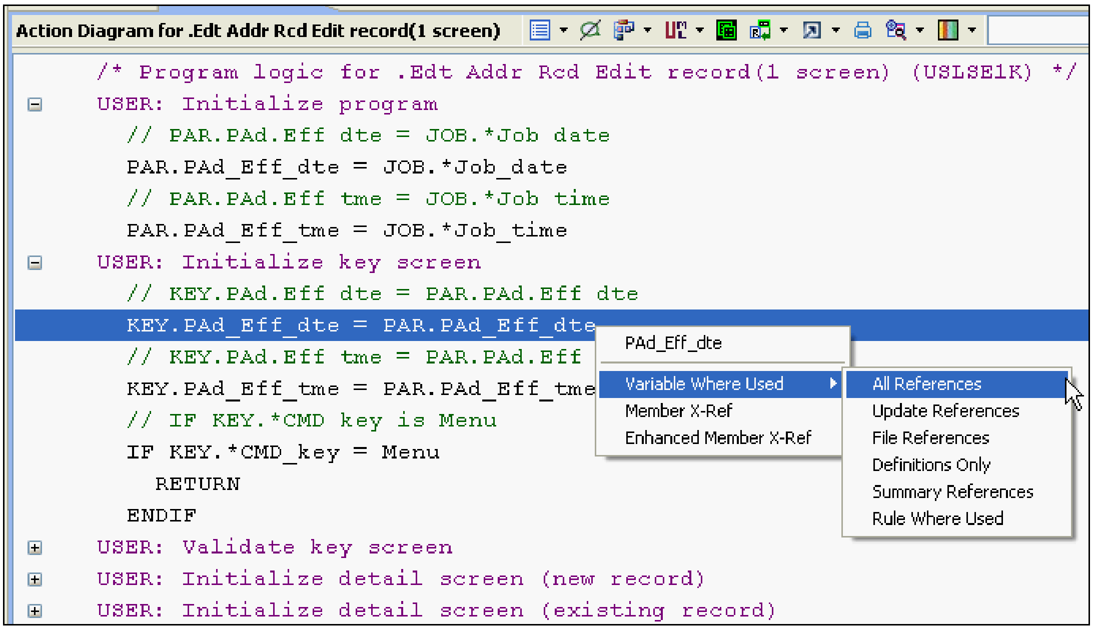
Action Diagram for .Edt Addr Rcd Edit record(1 screen)

```
  /* Program logic for .Edt Addr Rcd Edit record(1 screen) (USLSE1K) */
⊞ USER: Initialize program
⊞ USER: Initialize key screen
⊞ USER: Validate key screen
⊞ USER: Initialize detail screen (new record)
⊞ USER: Initialize detail screen (existing record)
⊞ USER: Delete DBF record
⊞ USER: Validate detail screen fields
⊞ CALC: Detail screen function fields
⊞ USER: Validate detail screen relations
⊞ USER: Create DBF record
⊞ USER: Change DBF record
⊞ USER: Process command keys
```

Expand the nodes of interest to see the underlying statements.

The user has an option to review the cross-reference information from the diagram. Right-click on the variable of interest and opt for Variable Where Used to see the list of statements using that variable.

Figure 23 – Looking at Cross-reference information

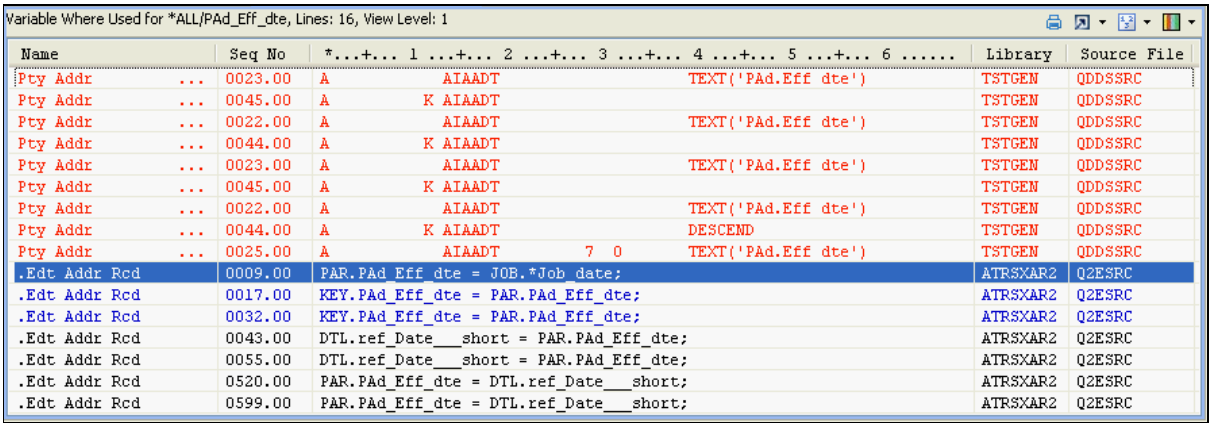


Action Diagram for .Edt Addr Rcd Edit record(1 screen)

```
/* Program logic for .Edt Addr Rcd Edit record(1 screen) (USLSE1K) */
USER: Initialize program
  // PAR.PAd.Eff dte = JOB.*Job date
  PAR.PAd_Eff_dte = JOB.*Job_date
  // PAR.PAd.Eff tme = JOB.*Job time
  PAR.PAd_Eff_tme = JOB.*Job_time
USER: Initialize key screen
  // KEY.PAd.Eff dte = PAR.PAd.Eff dte
  KEY.PAd_Eff_dte = PAR.PAd_Eff_dte
  // KEY.PAd.Eff tme = PAR.PAd.Eff
  KEY.PAd_Eff_tme = PAR.PAd_Eff_tme
  // IF KEY.*CMD key is Menu
  IF KEY.*CMD_key = Menu
    RETURN
  ENDIF
USER: Validate key screen
USER: Initialize detail screen (new record)
USER: Initialize detail screen (existing record)
```

The action brings up the list of statements for the actual variable and its mnemonic or short name. The screen below shows the statements from the Physical files and the programs. Note, the former are for the mnemonic or short name and the latter are for the actual variable or the long name.

Figure 24 – Cross-reference information for a variable



Variable Where Used for *ALL/PAd_Eff_dte, Lines: 16, View Level: 1

Name	Seq No	*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6	Library	Source File
Pty Addr ...	0023.00	A AIAADT TEXT('PAd.Eff dte')	TSTGEN	QDDSSRC
Pty Addr ...	0045.00	A K AIAADT	TSTGEN	QDDSSRC
Pty Addr ...	0022.00	A AIAADT TEXT('PAd.Eff dte')	TSTGEN	QDDSSRC
Pty Addr ...	0044.00	A K AIAADT	TSTGEN	QDDSSRC
Pty Addr ...	0023.00	A AIAADT TEXT('PAd.Eff dte')	TSTGEN	QDDSSRC
Pty Addr ...	0045.00	A K AIAADT	TSTGEN	QDDSSRC
Pty Addr ...	0022.00	A AIAADT TEXT('PAd.Eff dte')	TSTGEN	QDDSSRC
Pty Addr ...	0044.00	A K AIAADT DESCEND	TSTGEN	QDDSSRC
Pty Addr ...	0025.00	A AIAADT 7 0 TEXT('PAd.Eff dte')	TSTGEN	QDDSSRC
.Edt Addr Rcd	0009.00	PAR.PAd_Eff_dte = JOB.*Job_date;	ATRSXAR2	Q2ESRC
.Edt Addr Rcd	0017.00	KEY.PAd_Eff_dte = PAR.PAd_Eff_dte;	ATRSXAR2	Q2ESRC
.Edt Addr Rcd	0032.00	KEY.PAd_Eff_dte = PAR.PAd_Eff_dte;	ATRSXAR2	Q2ESRC
.Edt Addr Rcd	0043.00	DTL.ref_Date___short = PAR.PAd_Eff_dte;	ATRSXAR2	Q2ESRC
.Edt Addr Rcd	0055.00	DTL.ref_Date___short = PAR.PAd_Eff_dte;	ATRSXAR2	Q2ESRC
.Edt Addr Rcd	0520.00	PAR.PAd_Eff_dte = DTL.ref_Date___short;	ATRSXAR2	Q2ESRC
.Edt Addr Rcd	0599.00	PAR.PAd_Eff_dte = DTL.ref_Date___short;	ATRSXAR2	Q2ESRC

The user can double-click on the desired line to reach the source of interest.

Using Design Recovery for Rebuilding

Whilst Design Recovery is very valuable for documentation and application support purposes the real benefits come when the recovered design can be used to modernize or re-develop a system. Reusing existing designs programmatically can provide a dramatic productivity gain in rebuilding an application. While legacy application designs in their entirety might not suit a modern application implementation, design components are often suitable, as long they can be re-used at a sufficiently high level without introducing complexity to the redeveloped application. Therefore, being able to select and enhance, or ignore these at a granular level, removes the inheritance of irrelevant or legacy-specific code constructs, and allows direct access to elements that might have otherwise been deemed unusable in their current form.

NB: This is an important point so worth stressing - this lets us bring across what is useful leaving behind what isn’t relevant and cuts down on duplication

Another important factor in this scenario is the ability to choose an implementation technology or language that suits the technology constraints or resources specific to a region or organization. The next sections cover how we can use the recovered application design in different ways to effect varying levels of modernization and re-development up to and including a rebuilt system.

Database Modernization - using the Data Model assets

Whilst it has always been possible to access System i data in a relational database like fashion there was originally no way of defining your database in the traditional relational database form with a schema or model. This has meant that most System i applications don't have an explicitly defined relational database schema or model.

The data model for a legacy application as deduced by X-Analysis can be used to modernize the database and database access as well as providing valuable information for analysis and documentation. Once you have a modernized database you gain a number of advantages:

1. Easier access to your data for reporting via Business Intelligence (BI) tools when they use the newly derived Data Model.

2. Ability to use modern Object Relational Mapping (ORM) software such as Hibernate for rapid application development in Java and other modern languages.

3. Because the database is defined in purely SQL terms rather than in a proprietary file format it becomes portable i.e. it is now an option to consider moving the database to another platform.

4. Openness and Standards compliance using Industry standard SQL means that many different tools and applications on multiple platforms can easily access and use your modernized database

5. Improved performance as IBM's data retrieval efforts have been concentrated on SQL access rather than file based access for many years now

6. Reduced dependency on System i specific skills such as DDS, which may led to cost savings and reduced risk.

7. Data Integrity - Journaling is available for SQL access just as it has always been for file-based access. Constraints and referential integrity can be implemented directly at the database level where they are unavoidable rather than at the program level. Databases triggers allow code to be run before or after records are added, updated or deleted providing an easy way of enforcing compliance, audits, validations and applying business rules.

We've now looked at some of the advantages of Database Modernization but how is it actually achieved and how can X-Analysis help the process along?

As previously mentioned historically System i applications have used Data Description Specifications (DDS) to define physical files and associated logical files or access paths. Whilst the files created can of course be accessed using SQL syntax from programs or via

JDBC/ODBC the actual definitions bear no relation to SQL. What the process of X-Analysis database modernization does is to replace the DDS definitions with SQL create scripts that build tables and indexes. As we have seen X-Analysis has a complete cross-reference of all files and fields and their relationships and can build the table creation scripts together with the required indexes to optimize the system, your existing programs all still work after this process and without any re-compilation or alteration.

By using X-Analysis to do this automatically, no existing programs need be recompiled or impacted in any way. With the data copying facilities built in to the tool, the transition can be seamless. From this point onwards legacy programs can continue to use native I/O techniques, or be automatically reengineered using Databorough's X-Modernize, to use externalized SQL I/O.

X-Analysis can also generate an entire set of CRUD[1] RPG stateless I/O modules to be used as web services for any web or SOA type development.

For IBM's take on the relative merits of DDS and SQL and the advantages of an SQL created database over one created with DDS see the IBM Redbook "Modernizing IBM eServer iSeries Application Data Access - A Roadmap Cornerstone" which is an excellent reference on this subject.

Modern Data I/O & Persistence – Modern Object Oriented (OO) type development does not mean that relational databases need be abandoned. Indeed, it would be a fairly unwise strategy for a company to throw away its database and the information stored in it, for the sake of OO development. Java is an Object Oriented or OO language. System i databases are relational. Though Java Database Connectivity or, JDBC provides an easy method for accessing relational databases, it is basically a low level API providing only a thin layer of abstraction. Thus complex I/O and data requirements typical of System i applications quickly become very complicated to develop and maintain. JDBC is sufficient for small and medium projects, but is not that well suited for enterprise level applications. Therefore what is required is an Object Relational Mapping (ORM) framework that can act as a mediator between an OO design and a relational database.

The most widely used ORM framework for Java is Hibernate. Hibernate (www.hibernate.org) is a free open source Java package originated and backed by JBoss and Red Hat that makes it easier for Java developers to work with relational databases as it handles what's known as the persistence layer i.e. the bit that actually reads and writes data to the database. Hibernate has been downloaded at least 3 million times and has gained widespread usage so there are now plenty of books and resources available for it.

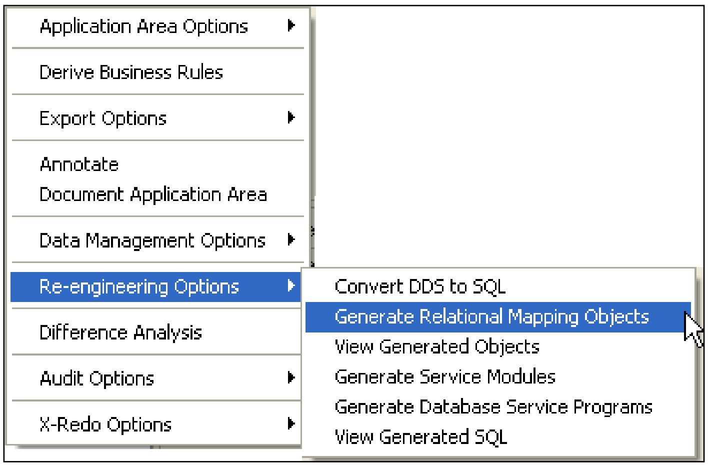
[1] CRUD Create, Read, Update, Delete - a tongue in cheek acronym coined to cover the key table operations any system has have.

Hibernate allows Java developers to treat the database as a set of objects like any other object they use, thus dramatically simplifying the code they need to write. For large complicated databases, typical of system i applications, this is naturally a big advantage. It is this, which makes Hibernate one of the most popular persistence frameworks used for enterprise Java applications today.

As a framework Hibernate naturally requires set up and configuration, and the more information that can be supplied in the configuration, the more effectively it can be utilized in development and production. Though DDL Schemas can be imported directly into Hibernate, DDL that describes only the tables and fields of the physical database is only part of the ORM requirement. Relational or foreign key information is obviously the next critical requirement for the Hibernate ORM to work effectively. As explained earlier, the entire legacy relational model derived by X-Analysis can be exported as DDL. This DDL can then be imported directly into Hibernate, thus producing an instant Object relational Map or ORM of the entire legacy application database.

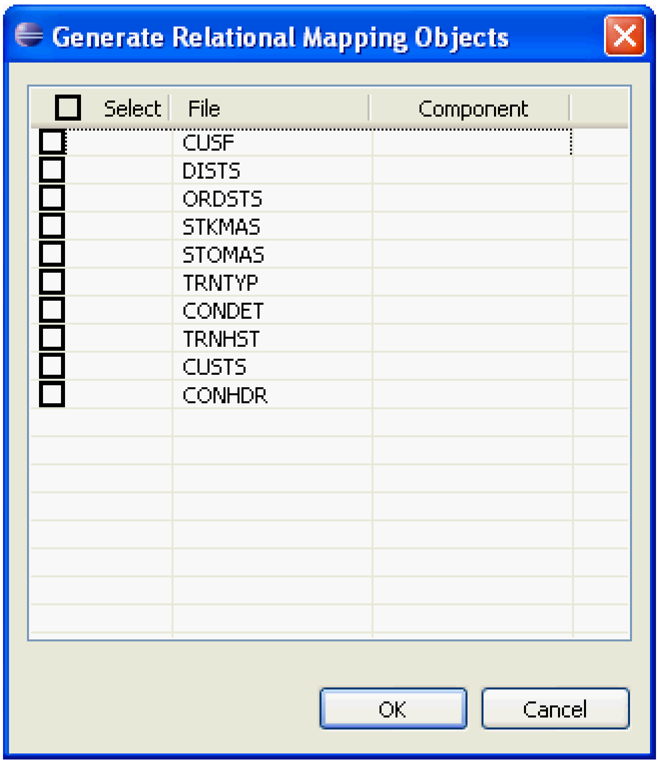
Hibernate is a very good fit for 2E applications as it is flexible enough to work well with the mnemonic naming without having to use the file/table names as with pure JDBC.

To illustrate Hibernate and X-Analysis lets look at generating ORM objects with X-Analysis this is accessed by a straight forward right click menu;



Choose the files you'd like to generate ORM objects for:

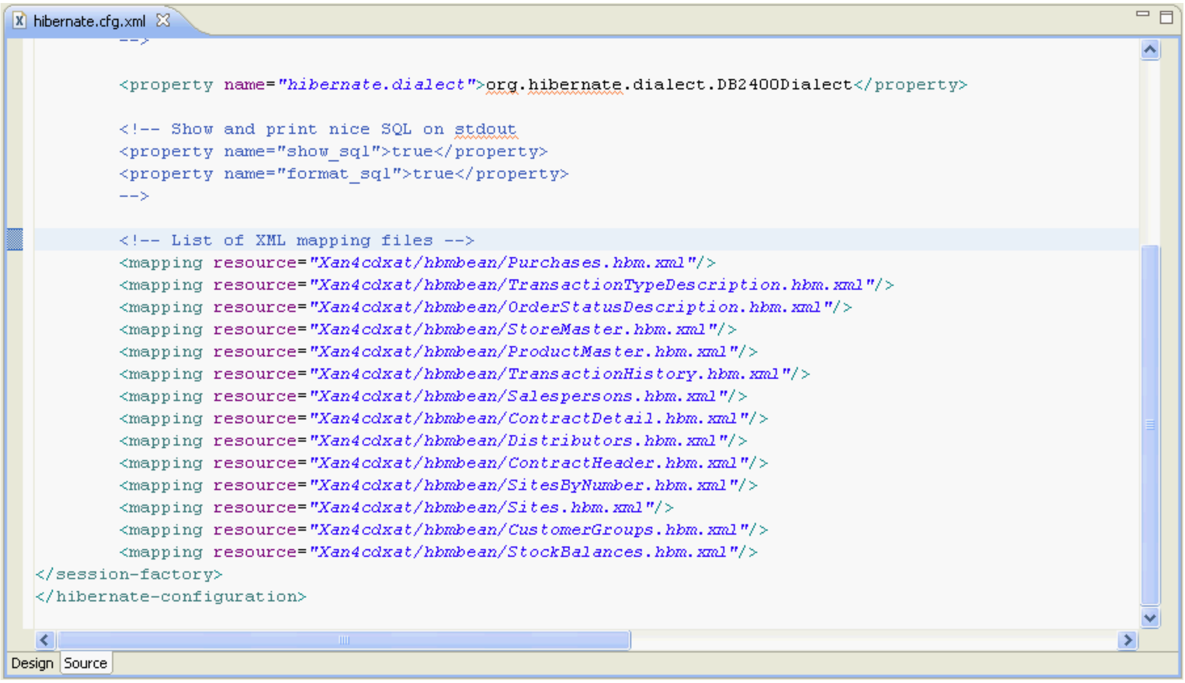
Figure 25 – Generate ORM Objects dialog



□ Select	File	Component
□	CUSF	
□	DISTS	
□	ORDSTS	
□	STKMAS	
□	STOMAS	
□	TRNTYP	
□	CONDET	
□	TRNHST	
□	CUSTS	
□	CONHDR	

X-Analysis generates hibernate mappings , configurations and hibernate java beans - everything required to access those tables from Java:

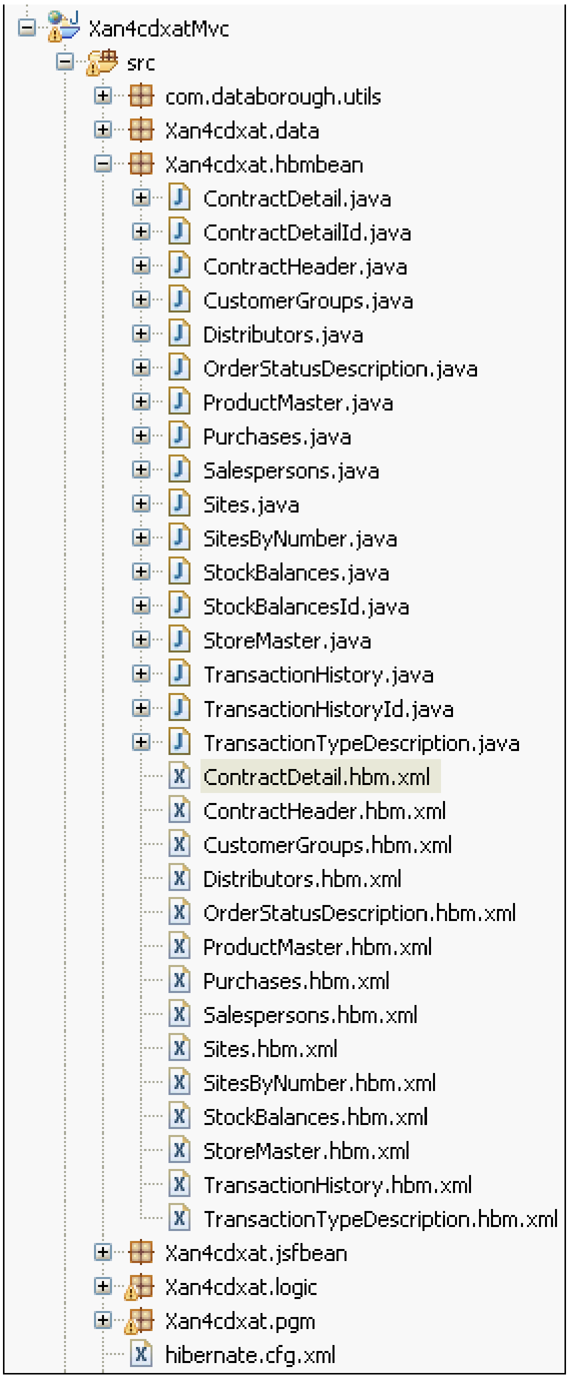
Figure 26 – ORM relational mappings for tables



```
        -->

        <property name="hibernate.dialect">org.hibernate.dialect.DB2400Dialect</property>

        <!-- Show and print nice SQL on stdout
        <property name="show_sql">true</property>
        <property name="format_sql">true</property>
        -->

        <!-- List of XML mapping files -->
        <mapping resource="Xan4cdxat/hbmbean/Purchases.hbm.xml"/>
        <mapping resource="Xan4cdxat/hbmbean/TransactionTypeDescription.hbm.xml"/>
        <mapping resource="Xan4cdxat/hbmbean/OrderStatusDescription.hbm.xml"/>
        <mapping resource="Xan4cdxat/hbmbean/StoreMaster.hbm.xml"/>
        <mapping resource="Xan4cdxat/hbmbean/ProductMaster.hbm.xml"/>
        <mapping resource="Xan4cdxat/hbmbean/TransactionHistory.hbm.xml"/>
        <mapping resource="Xan4cdxat/hbmbean/Salespersons.hbm.xml"/>
        <mapping resource="Xan4cdxat/hbmbean/ContractDetail.hbm.xml"/>
        <mapping resource="Xan4cdxat/hbmbean/Distributors.hbm.xml"/>
        <mapping resource="Xan4cdxat/hbmbean/ContractHeader.hbm.xml"/>
        <mapping resource="Xan4cdxat/hbmbean/SitesByNumber.hbm.xml"/>
        <mapping resource="Xan4cdxat/hbmbean/Sites.hbm.xml"/>
        <mapping resource="Xan4cdxat/hbmbean/CustomerGroups.hbm.xml"/>
        <mapping resource="Xan4cdxat/hbmbean/StockBalances.hbm.xml"/>
</session-factory>
</hibernate-configuration>
```

Generated hibernate xml mappings and hibernate java beans are as shown below:

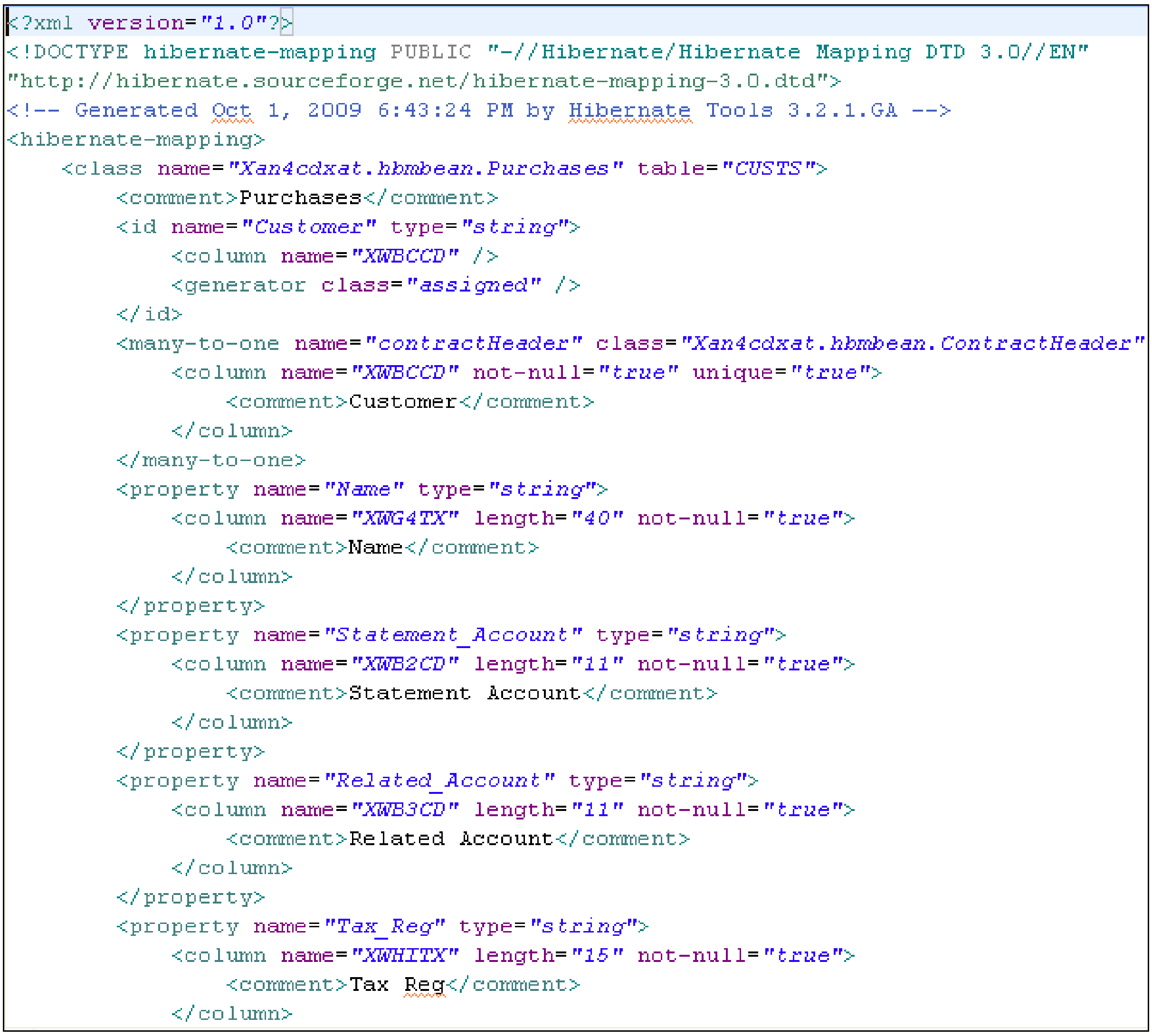
Figure 27 – Generated hibernate Xml mappings and Java beans



```
Xan4cdxatMvc
  src
    com.databorough.utils
    Xan4cdxat.data
    Xan4cdxat.hbmbean
      ContractDetail.java
      ContractDetailId.java
      ContractHeader.java
      CustomerGroups.java
      Distributors.java
      OrderStatusDescription.java
      ProductMaster.java
      Purchases.java
      Salespersons.java
      Sites.java
      SitesByNumber.java
      StockBalances.java
      StockBalancesId.java
      StoreMaster.java
      TransactionHistory.java
      TransactionHistoryId.java
      TransactionTypeDescription.java
      ContractDetail.hbm.xml
      ContractHeader.hbm.xml
      CustomerGroups.hbm.xml
      Distributors.hbm.xml
      OrderStatusDescription.hbm.xml
      ProductMaster.hbm.xml
      Purchases.hbm.xml
      Salespersons.hbm.xml
      Sites.hbm.xml
      SitesByNumber.hbm.xml
      StockBalances.hbm.xml
      StoreMaster.hbm.xml
      TransactionHistory.hbm.xml
      TransactionTypeDescription.hbm.xml
    Xan4cdxat.jsfbean
    Xan4cdxat.logic
    Xan4cdxat.pgm
    hibernate.cfg.xml
```

Hibernate mapping for a table looks like:

Figure 28 – Hibernate mapping for a table



```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated Oct 1, 2009 6:43:24 PM by Hibernate Tools 3.2.1.GA -->
<hibernate-mapping>
    <class name="Xan4cdxat.hbmbean.Purchases" table="CUSTS">
        <comment>Purchases</comment>
        <id name="Customer" type="string">
            <column name="XWBCCD" />
            <generator class="assigned" />
        </id>
        <many-to-one name="contractHeader" class="Xan4cdxat.hbmbean.ContractHeader"
            <column name="XWBCCD" not-null="true" unique="true">
                <comment>Customer</comment>
            </column>
        </many-to-one>
        <property name="Name" type="string">
            <column name="XWG4TX" length="40" not-null="true">
                <comment>Name</comment>
            </column>
        </property>
        <property name="Statement_Account" type="string">
            <column name="XWB2CD" length="11" not-null="true">
                <comment>Statement Account</comment>
            </column>
        </property>
        <property name="Related_Account" type="string">
            <column name="XWB3CD" length="11" not-null="true">
                <comment>Related Account</comment>
            </column>
        </property>
        <property name="Tax_Reg" type="string">
            <column name="XWHITX" length="15" not-null="true">
                <comment>Tax Reg</comment>
            </column>
```

Modern Business Intelligence, Inquiries & Reports – There are many choices when it comes to BI or reporting tools. Almost all of these automatically allow import of the database definition, some provide for the relational or foreign key model of a database, some even try and infer this from the physical implementation of the data base fields and file definitions themselves. The problem with using the file and field names on a System i database, is that they do not match, and they have complex key structures as opposed to UID or sequence keys. The benefits of having an explicitly defined relational and physical model of the database in any of these tools are significant. Almost all reports use some form of file joining information for displaying code descriptions or related information such as list prices on an order for example. Drill-down applications use join or foreign key information to build the navigation links into the reports or queries. The DDL export of the relational and physical model derived by X-Analysis provides a distinct productivity advantage to anyone building reports or BI Applications with any of these tools. It is also possible to populate and build an entire BI application from the data model. The application design is stored as meta-data, and then can be generated into reporting tools such as DB2 Web-Query.

Rebuilding the View

We had seen above how useful screen design information is extracted into Function Definitions in the X-Analysis repository. These function definitions are effectively input specifications for generating new UI's or Views. The Modernization Tool Set of X-Analysis actually uses the Function Definitions to automatically, generate JSF/Facelets and Java bean source for each recovered screen design. The generated source code is structured, annotated, simple, industry standard, and ready for maintenance with any Java IDE. Layouts and styles are implemented using CSS, and certain field types implemented with appropriate HTML controls such as date prompts, drop downs, check boxes with corresponding Java Scripting or logic in the Java Bean.

EGL versions are also available for view/controller generation, with future generation options for PHP, C#, and XAML, becoming available from Databorough and other Business Partners.

Actions from the Function definitions translate effectively into links on the generated JSF/Facelets, and these can be implemented with tab, buttons, or any appropriate UI standard demanded by a project. The required logic to invoke these actions is placed in the appropriate methods of the generated Java Bean as described below.

Rebuilding the Controller

The Java bean that drives the JSF/Facelet has standard methods for Data I/O, navigational actions, and for using any residual services that might remain on the legacy server in RPG, COBOL or 2E. This JSF bean has standardized exit points and a set of standard parameters making maintenance and development more efficient and consistent.

A separate call bean is implemented for each transaction group or legacy service program. This call bean provides a standard interface to these reengineered legacy RPG services, and therefore greatly simplifies the controller or JSF bean as it is often referred to. In the case of a set multiple JSF's that make up a transaction, the call bean also acts as a persistence manager for the transaction.

Reusing Business Rules

The optimum design objective is to move as much of the business rule logic into the Java as possible, thus reducing the dependency on legacy languages. The monolithic architecture of legacy applications produces significant amounts of redundant and duplicate validation and field or calculation logic type business rules. These need to be re-factored if the maintainability of the application maintenance is to be improved – a primary objective of modernization in the first place. Examples of this refactoring process are:

- Centralizing commonly used referential validation rules into the data persistence framework.
- Date formatting logic can be centralized into reusable classes.
- Centralizing commonly used field logic, such as, global tax calculations into reusable classes.

This means that code duplication & redundancy can be almost completely avoided in the modern application. Typically, the only logic recreated in UI specific classes or beans will be context specific calculations field logic such as calculating the value of the order line being captured, along with conditional display or navigation logic for some UI's These new beans/classes should therefore be fairly simple and easy to maintain by comparison to their legacy counterparts.

X-Analysis provides powerful features as described earlier for narrating, annotating, and carrying out where-used analysis on Business Rules. These functions combined with the ability to interactively select those rules from the legacy application for reuse in the modern version, dramatically speed up the refactoring process. Rules selected by the user are then placed automatically into the Java bean, with the Pseudo code narrative describing the rule included as annotations in the bean.

The fact that we can easily verify that business rules from the legacy system have been built into the new system provides a high degree of confidence in the new system and is important from a compliance and audit standpoint.

Each legacy program can also be reengineered into a standardized RPG service program as a stored procedure for handling legacy batch services that need not be touched, either temporarily or indefinitely. A Java call bean is created automatically by X-Analysis in this instance, and acts as a wrapper class to this stored procedure, thus greatly simplifying access to legacy services for Java developers. Conversion to EGL is another option available for residual RPG based service logic.

Summary

Comprehensive, accurate, and current documentation of a legacy application improves quality, productivity and reduces risk, for any maintenance, modernization or rebuild IT project. The risk associated with maintaining large complex legacy application, with a rapidly diminishing set of legacy skills, can be largely mitigated by access to such documentation.

Understanding and mapping the relevance of existing designs, and quantifying the scope and metrics of an application, is the first step in ANY modernization project, even if the application is to be replaced. Design constructs such as the data model can be used for support, development, and testing tasks such as test data extraction. Source change management can be vastly improved by powerful cross-referencing functionality.

Legacy design constructs can be used passively in the form of information they represent, and programmatically to radically accelerate application rebuilds; a requirement for achieving true long-term application modernization. A combination of both allows optimum use of internal and external resources and existing design assets.

X-Analysis delivers against all of these concepts. 20+ years of development effort, ensures that virtually any legacy application can be automatically reverse engineered onto a high-level design.

Additional Resources & Information

These are the few relevant references used in this white paper. More information is available from the authors of this white paper upon request.

World Leader in discovery, analysis and modernization tools for System i

http://www.databorough.com

Modernizing and Improving the Maintainability of RPG Applications Using X-Analysis Version 5.6 – IBM Redbook

http://www.redbooks.ibm.com/redpieces/abstracts/redp4046.html

Architecture and Design Recovery - Johannes Kepler University

http://www.alexander-egyed.com/research/software_architecture_and_design_recovery.html

Crafting an Application Architecture with Java Frameworks - by Don Denoncourt

http://systeminetwork.com/article/crafting-application-architecture-java-frameworks

Hibernate Your JDBC - by Don Denoncourt

http://systeminetwork.com/article/hibernate-your-jdbc

Encore's Extreme Makeover from the Inside Out - by Richard Shaler

http://systeminetwork.com/article/encores-extreme-makeover-inside-out

A Field Guide to Encore's System i Software – by Richard Shaler

http://systeminetwork.com/article/field-guide-encores-system-i-software

Asset modernization: Discover and transform legacy assets for reuse – IBM Rational

http://www-306.ibm.com/software/info/developer/solutions/em/systems/i/assets/index.jsp

Modernizing IBM eServer iSeries Application Data Access - A Roadmap Cornerstone – IBM Redbook

http://www.redbooks.ibm.com/redbooks/SG246393/wwhelp/wwhimpl/java/html/wwhelp.htm

Eating the IT Elephant - Moving from Greenfield Development to Brownfield - by Richard Hopkins and Kevin Jenkins IBM Press http://eatingtheitelephant.com/home.html